LIBRARY ROUTINES
(*SY2)

by

Jeanne Adams and Paul Rotar

Computing Facility
National Center for Atmospheric Research*
Boulder, Colorado 80302

December 1969

*The National Center for Atmospheric Research is sponsored by the National Science Foundation.
LIBRARY ROUTINES
(*SY2)

by

Jeanne Adams and Paul Rotar

Computing Facility
National Center for Atmospheric Research*
Boulder, Colorado 80302

December 1969

*The National Center for Atmospheric Research is sponsored by the National Science Foundation.
ACKNOWLEDGMENTS

The authors would like to thank Bernard O'Lear for his contribution to the writeups on subroutines BUFRD, OVERLAY, RANRD, and SAVE; David Robertson for writing the section on Plotting Routines; Roy Jenne and Dennis Joseph for their contributions to the writeups on subroutines GBYTE and IOPROC and the section on Routines to Process and Access Data; and Paul Swarztrauber and Gerald Browning for their contributions to the section on Special Purpose Subroutines.
Library routines are available to the programmer from several sources. Some of these subroutines (or functions) are in the library and are loaded at object time if the program calls them. The macro algorithms are edited directly into the program by the compiler and are referred to as "in-line" routines.

Certain mathematical routines, coded by NCAR programmers in Fortran or Ascent, are available from our staff librarians and must be submitted as subprograms. These are described in the section "Special Purpose Subroutines."

In addition, NCAR programmers have written special routines which are closely related to scientific problems in meteorology. These are discussed in the section "Routines to Process and Access Data."

The "Plotting Routines" section discusses the subroutines available for generating graphical and printed output on microfilm.
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-line functions</td>
<td>100</td>
</tr>
<tr>
<td>General description</td>
<td>100</td>
</tr>
<tr>
<td>Table of in-line algorithms</td>
<td>101</td>
</tr>
<tr>
<td>Mathematical library routines</td>
<td>200</td>
</tr>
<tr>
<td>General description</td>
<td>200</td>
</tr>
<tr>
<td>Call by value</td>
<td>200</td>
</tr>
<tr>
<td>Call by name</td>
<td>202</td>
</tr>
<tr>
<td>Table of Fortran library functions</td>
<td>205</td>
</tr>
<tr>
<td>Writeups of mathematical library routines</td>
<td></td>
</tr>
<tr>
<td>ACOS</td>
<td>207</td>
</tr>
<tr>
<td>ALOG</td>
<td>208</td>
</tr>
<tr>
<td>ALOG10</td>
<td>209</td>
</tr>
<tr>
<td>AMAX/AMIN</td>
<td>210</td>
</tr>
<tr>
<td>ASIN/ACOS</td>
<td>212</td>
</tr>
<tr>
<td>ATAN</td>
<td>213</td>
</tr>
<tr>
<td>ATAN2</td>
<td>215</td>
</tr>
<tr>
<td>CABS</td>
<td>217</td>
</tr>
<tr>
<td>CBAIEX</td>
<td>218</td>
</tr>
<tr>
<td>CCOS</td>
<td>219</td>
</tr>
<tr>
<td>CEXP</td>
<td>220</td>
</tr>
<tr>
<td>CLOG</td>
<td>221</td>
</tr>
<tr>
<td>COS</td>
<td>222</td>
</tr>
<tr>
<td>CSIN</td>
<td>223</td>
</tr>
<tr>
<td>CSQRT</td>
<td>224</td>
</tr>
<tr>
<td>DATAN</td>
<td>225</td>
</tr>
<tr>
<td>DATAN2</td>
<td>226</td>
</tr>
<tr>
<td>DBADEX</td>
<td>228</td>
</tr>
<tr>
<td>DBAIEX</td>
<td>229</td>
</tr>
<tr>
<td>DBAREX</td>
<td>230</td>
</tr>
<tr>
<td>DCOS</td>
<td>231</td>
</tr>
<tr>
<td>DEXP</td>
<td>232</td>
</tr>
<tr>
<td>DLOG</td>
<td>234</td>
</tr>
<tr>
<td>DLOG10</td>
<td>236</td>
</tr>
<tr>
<td>DMAX1/DMIN1</td>
<td>237</td>
</tr>
<tr>
<td>DMOD</td>
<td>238</td>
</tr>
<tr>
<td>DSIGN</td>
<td>239</td>
</tr>
<tr>
<td>DSIN</td>
<td>240</td>
</tr>
<tr>
<td>DSQRT</td>
<td>241</td>
</tr>
<tr>
<td>EXP</td>
<td>242</td>
</tr>
<tr>
<td>IBAIEX</td>
<td>244</td>
</tr>
<tr>
<td>IDINT</td>
<td>245</td>
</tr>
<tr>
<td>RANF</td>
<td>246</td>
</tr>
<tr>
<td>RBAIEX</td>
<td>248</td>
</tr>
<tr>
<td>RBAREX</td>
<td>249</td>
</tr>
<tr>
<td>SIN/COS</td>
<td>250</td>
</tr>
<tr>
<td>SNGL</td>
<td>251</td>
</tr>
<tr>
<td>Mathematical library routines (cont'd)</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>--</td>
</tr>
<tr>
<td>Writeups of mathematical library routines (cont'd)</td>
<td></td>
</tr>
<tr>
<td>SORT</td>
<td>252</td>
</tr>
<tr>
<td>TAN</td>
<td>253</td>
</tr>
<tr>
<td>TANH</td>
<td>254</td>
</tr>
<tr>
<td>TIMEF</td>
<td>255</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Utility library subroutines</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>General description</td>
<td>300</td>
</tr>
<tr>
<td>Writeups of utility routines:</td>
<td></td>
</tr>
<tr>
<td>ACGOER</td>
<td>302</td>
</tr>
<tr>
<td>BACKSP</td>
<td>303</td>
</tr>
<tr>
<td>BUFFEI</td>
<td>304</td>
</tr>
<tr>
<td>BUFRD</td>
<td>307</td>
</tr>
<tr>
<td>DEBUG</td>
<td>315</td>
</tr>
<tr>
<td>DUMP</td>
<td>319</td>
</tr>
<tr>
<td>ENDFIL</td>
<td>320</td>
</tr>
<tr>
<td>EXIT</td>
<td>321</td>
</tr>
<tr>
<td>GBYTE</td>
<td>322</td>
</tr>
<tr>
<td>INPUTB</td>
<td>326</td>
</tr>
<tr>
<td>INPUTC</td>
<td>328</td>
</tr>
<tr>
<td>INPUTS</td>
<td>330</td>
</tr>
<tr>
<td>IOPROC</td>
<td>332</td>
</tr>
<tr>
<td>KODER</td>
<td>336</td>
</tr>
<tr>
<td>KRAKER</td>
<td>337</td>
</tr>
<tr>
<td>OUTPTC</td>
<td>338</td>
</tr>
<tr>
<td>OUTPTS</td>
<td>341</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>342</td>
</tr>
<tr>
<td>PAUSE</td>
<td>352</td>
</tr>
<tr>
<td>PDUMP</td>
<td>353</td>
</tr>
<tr>
<td>RANRD</td>
<td>356</td>
</tr>
<tr>
<td>SAVEF</td>
<td>360</td>
</tr>
<tr>
<td>SORT</td>
<td>366</td>
</tr>
<tr>
<td>SSW</td>
<td>368</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plotting routines</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>400</td>
</tr>
<tr>
<td>PLOT</td>
<td>401</td>
</tr>
<tr>
<td>The dd80</td>
<td>401</td>
</tr>
<tr>
<td>Scaling</td>
<td>402</td>
</tr>
<tr>
<td>SET</td>
<td></td>
</tr>
<tr>
<td>PSSCALE</td>
<td></td>
</tr>
<tr>
<td>PORGN</td>
<td></td>
</tr>
<tr>
<td>Plotting</td>
<td>406</td>
</tr>
<tr>
<td>FRSTPT</td>
<td></td>
</tr>
<tr>
<td>VECTOR</td>
<td></td>
</tr>
<tr>
<td>POINT</td>
<td></td>
</tr>
<tr>
<td>PSYM</td>
<td></td>
</tr>
<tr>
<td>LINE</td>
<td></td>
</tr>
<tr>
<td>CURVE</td>
<td></td>
</tr>
</tbody>
</table>
Plotting routines (cont'd)
Writing ............................................. 410
PWRT
Control .............................................. 415
Plot buffer control ................................... 415
FLUSH
FRAME
Other control routines ................................ 417
OPTION
DASHLN
MXMY
Backgrounds .......................................... 421
GRID
PERIM
PERIML
GRIDL
HALFAX
AXES
GRIDALL
TICKS
TICK4
LABMOD
Plot instruction manipulation .......................... 431
FLASH1
FLASH2
FLASH3
FLASH4
Instruction format .................................... 436
Other useful routines .................................. 438
CALCNT
IDIOT
SUPMAP
Special purpose subroutines ............................ 500
Algebraic zeros ....................................... 501
CUBIC
PROOT
TRAUB
Approximation and curve fitting ....................... 501
LINLSQ
LSTSQR
NOLLS
RATION
CSP
SPLINE
Elementary functions ................................. 502
EI
ERF
BESLJ
BESLY
BESIK
BESLI
EXP
Special purpose subroutines (cont'd)

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFA</td>
<td>Fast Fourier transforms</td>
<td>502</td>
</tr>
<tr>
<td>FFS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOUR2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FOURT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphic display</td>
<td></td>
<td>503</td>
</tr>
<tr>
<td>CALCNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDIOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input/output</td>
<td></td>
<td>503</td>
</tr>
<tr>
<td>RECIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matrix analysis</td>
<td></td>
<td>503</td>
</tr>
<tr>
<td>CMPXQR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REALQR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RSYMQR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INVMIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORTHO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORTHO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HESREL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HESREM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miscellaneous</td>
<td></td>
<td>505</td>
</tr>
<tr>
<td>NDATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PERMUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RENO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-algebraic zeros</td>
<td></td>
<td>505</td>
</tr>
<tr>
<td>LAGUER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLEQS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTSRH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ordinary differential equations</td>
<td></td>
<td>505</td>
</tr>
<tr>
<td>EZNDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNGKUT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ODEIG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partial differential equations</td>
<td></td>
<td>506</td>
</tr>
<tr>
<td>ADI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadrature</td>
<td></td>
<td>506</td>
</tr>
<tr>
<td>ALGUER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAUS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROMB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPNE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMPSON</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HASELI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random number generators</td>
<td></td>
<td>506</td>
</tr>
<tr>
<td>RANF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RNDEV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statistics</td>
<td></td>
<td>507</td>
</tr>
<tr>
<td>ASA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMD04M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMD06M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMD02R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMD05R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BMD02S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table of Contents

### Routines to process and access data
- Tape copy and edit ........................................ 600
- User subroutines ........................................ 600
  - USER
  - USERF
- Control cards ........................................... 602
- Copy output ............................................. 604
- RPTOUT and RPTIN ........................................ 606
  - Write logical record ................................. 607
  - Read logical record ................................. 608
- Other data handling routines ......................... 610
  - RDTAPE/WRTAPE CHCONV
  - SORTUU/SORTBB CDS360
  - GBYTES/SBYTES REPBIN
  - GBITS/SBITS CORFOR
  - UBLOK/UZBLOK DREC
  - CDTAPE/CDTAPEB

### Central monitor ........................................ 700
- Storage ................................................. 700
- Error messages .......................................... 700
- Purpose ................................................... 700
- Registers destroyed .................................... 700
- Calling sequence ....................................... 700
- Input/output file handling ............................. 705
- Miscellaneous .......................................... 706
- Examples .................................................. 707

### Execution diagnostics ................................. 800
- General discussion ....................................... 800
- Mathematical library functions ....................... 802
- Utility routines ......................................... 803

---

### Library Routines

- RDTAPE/WRTAPE CHCONV
- SORTUU/SORTBB CDS360
- GBYTES/SBYTES REPBIN
- GBITS/SBITS CORFOR
- UBLOK/UZBLOK DREC
- CDTAPE/CDTAPEB
IN-LINE FUNCTIONS

The code for in-line functions is placed directly in the user's program by the compiler. These are available only through a Fortran reference. The logic is inserted in the object code of the Fortran program.

The following is an example of the compilation of the statement:

\[ E = \text{FLOAT}(J) \]

The compiler inserts in Ascent:

- \text{SA4} \quad J \quad \text{(bring J in from core)}
- \text{PX7} \quad B0,X4 \quad \text{(pack the bias 2000)}
- \text{NX6} \quad B0,X7 \quad \text{(normalize)}
- \text{SA6} \quad E \quad \text{(store in E)}

The operation is to take the value of the integer J and place it in E as a normalized floating point number.

In-line function names do not appear in the loader map. Programmer defined routines with the same name as in-line routines will be loaded and not called. The in-line routine will be compiled instead. The following table lists the in-line routines available on the NCAR compiler. Fortran II names appear in parentheses.
## In-Line Algorithms

<table>
<thead>
<tr>
<th>Description</th>
<th>Forms</th>
<th>Mode of Parameters</th>
<th>Mode of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Absolute Value</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain absolute value</td>
<td>ABS(X)</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td></td>
<td>(ABSF(X))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain absolute value</td>
<td>IABS(I)</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>(XABSF(I))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain absolute value</td>
<td>DABS(D)</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td><strong>Conversion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Truncate real argument</td>
<td>AINT(X)</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td></td>
<td>(INTF(X))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To truncate a real number</td>
<td>INT(X)</td>
<td>real</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>(XINTF(X))</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFIX(X)</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>(XFIXF(X))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To float an integer number</td>
<td>FLOAT(I)</td>
<td>integer</td>
<td>real</td>
</tr>
<tr>
<td>to floating point</td>
<td>(FLOATF(I))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obtain imaginary part of a complex argument</td>
<td>AIMAG(C)</td>
<td>complex</td>
<td>real</td>
</tr>
<tr>
<td>Obtain a double-precision number</td>
<td>DBLE(C)</td>
<td>complex</td>
<td>double</td>
</tr>
<tr>
<td>from real part of a complex number</td>
<td>REAL(C)</td>
<td>complex</td>
<td>real</td>
</tr>
<tr>
<td>Obtain real part of a complex argument</td>
<td>CMPLX(X₁,X₂)</td>
<td>real</td>
<td>complex</td>
</tr>
<tr>
<td>Convert real to complex</td>
<td>COMPLX(X₁,X₂)</td>
<td>real</td>
<td>complex</td>
</tr>
<tr>
<td>Complex conjugate of argument</td>
<td>CONJG(C)</td>
<td>complex</td>
<td>complex</td>
</tr>
<tr>
<td><strong>Modulus</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(X₁ - (X₁/X₂)X₂)</td>
<td>AMOD(X₁,X₂)</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>where</td>
<td>(MODF(X₁,X₂))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(X₁/X₂) = integer part</td>
<td>MOD(I₁,I₂)</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>(XMODF(I₁,I₂))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(cont'd)
In-Line Algorithms (cont'd)

<table>
<thead>
<tr>
<th>Description</th>
<th>Forms</th>
<th>Mode of Parameters</th>
<th>Mode of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign</td>
<td>SIGN(X₁, X₂)</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td></td>
<td>(SIGNF(X₁, X₂))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign of X₂ times absolute</td>
<td>ISIGN(I₁, I₂)</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>value of X₁</td>
<td>(XSIGNF(I₁, I₂))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>ALOC(VAR₁)</td>
<td>not applicable</td>
<td>real</td>
</tr>
<tr>
<td></td>
<td>(LOCF(VAR₁))</td>
<td></td>
<td>applicable</td>
</tr>
<tr>
<td></td>
<td>LOC(VAR₁)</td>
<td>not applicable</td>
<td>integer</td>
</tr>
<tr>
<td></td>
<td>(XLOCF(VAR₁))</td>
<td></td>
<td>applicable</td>
</tr>
</tbody>
</table>
MATHEMATICAL LIBRARY ROUTINES

GENERAL DESCRIPTION

Mathematical functions and subprograms that are used frequently by programmers are stored in the library and need not be submitted with the program (job). A reference to these will load the library program from the system along with the other routines submitted. A library function reference will contain at least one parameter. These library routines may be a "call by value" or a "call by name."

CALL BY VALUE

A call by value is used primarily for library functions with one argument. These function type routines return a value associated with the function name where the mode of the function is determined by a type indicator or the name of the function. The function call may be included anywhere in an arithmetic statement. For example, \( E = (4 \cdot X)/\text{ALG}(X) \) is an arithmetic statement where the value of \( \text{ALG}(X) \) is returned in an X-register for inclusion in the expression. When \( \text{ALG} \) is called in this way, the "value of X" is placed in an X-register on entry to the subprogram. "Values" of parameters are sent to the subprogram and returned by the subprogram via X-registers. Function subprograms on the library tape save all B-registers and restore them before returning to the main code.
CALL BY VALUE (cont'd) To illustrate the type of code generated by the compiler, an example of a library function call which is a call by value would be

\[
E = \text{ALG}(X)
\]

\[
\begin{align*}
\text{SA1} & \quad X \\
\text{RJ} & \quad \text{LOGF} \\
\text{BX6} & \quad X_1 \\
\text{SA6} & \quad E
\end{align*}
\]

The program would establish the address of the argument X and place the contents of X in register X1 by setting A1 to X. A return jump to the function is generated. The logarithm function code as a separate routine is loaded into core along with the other programs and subprograms used. The value associated with the logarithm of X is stored in X1 in the subprogram, and a jump is made back to the calling program. The value in X1 is transferred to X6 and stored in E by setting the address register A6 to the address of E.

Both Fortran II (\text{LOGF}) and Fortran IV (\text{ALG}) names are acceptable. The compiler may use Fortran II names in the RJ whenever Fortran II names have been defined. Both names (\text{LOGF} and \text{ALG}) appear in the loader map.

If a double-precision or complex routine is called and two values are returned, these are placed in X1 and X2.
CALL BY VALUE
(cont'd)

Example:

\[ z = \text{cc0s}(C) \]

| SA1 | C  |
| SA2 | C+1 |
| RJ  | \text{cc0s} |
| BX6 | X1  |
| BX7 | X2  |
| SA6 | Z   |
| SA7 | A6+1B |

The contents of C, the complex argument stored in C and C+1, are placed in X1 and X2. After the return jump, the value of the real part of the answer is in X1; the imaginary part is in X2. Z is type complex. The real part in X1 is placed in X6 and stored in Z by placing the address of Z in A6. The imaginary part of Z in X2 is placed in X7 and stored by setting A7 to the address of Z (in A6) plus 1. Similar code is generated for double-precision routines where the most significant part of the number returns in X1 and the least significant part in X2.

CALL BY NAME

Library routines may also be a "call by name." Addresses or names of parameters are sent to the subprogram. The subprogram must then set an A-register to the name in order to fetch the value of A in an X-register. Functions which are call by name return values according to the rules for functions stated above where results are returned by value in an X-register. Subroutines only return values by name as they appear in the subroutine call.
To illustrate the type of code generated by the compiler for a function call by name, the code compiled for \( A = X^{**}Y \) would be

\[
\begin{align*}
&\text{RJ} & \text{RBAREX} \\
&\text{EQ} & \times+3 \\
&\text{CON} & X \\
&\text{CON} & Y \\
&\text{BX6} & X1 \\
&\text{SA6} & A
\end{align*}
\]

The function would locate the addresses of \( X \) and \( Y \) in the two locations following the call. Then the function would set an A-register to the address when the contents of the address are needed.

All routines written in Fortran and supplied by the programmer are compiled as "call by name" subprograms. Function type subprograms must save and restore any B-registers used in the function, as the compiler assumes these will not be changed by a function call. Function subprograms return a value for the name of the function in X1. The compiler does not require B-register restoration by subroutines. If a user subprogram has the same name as a library routine, the user program will be loaded and used.

A table of these library functions is included for reference. The routines are described in detail following the table. They are listed alphabetically. All error messages printed from these subroutines are a result of a jump to SYSERR. The message is printed and execution is terminated. Fortran II names appear in parentheses.
Where the METHOD for each mathematical library routine is described, an asterisk (*) indicates that the routine on the system library has been checked for the given method. If no asterisk appears, the method stated is copied from Control Data literature as the method used in the routine supplied by Control Data Corporation.
### Fortran Library Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Parameter Type</th>
<th>Mode of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS (X)</td>
<td>Arccosine</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>ALOG (X)</td>
<td>Natural log of X</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>ALOG10 (X)</td>
<td>Log to the base 10 of X</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>AMAX0(I1, I2, ..., IN)</td>
<td>Maximum of n arguments*</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>AMIN0(I1, I2, ..., IN)</td>
<td>Minimum of n arguments*</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>ASIN (X)</td>
<td>Arcsine</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>ATAN (X)</td>
<td>Arctangent X radians</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>ATAN2(X1, X2)</td>
<td>Arctangent X1/X2</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>CABS (C)</td>
<td>Absolute value</td>
<td>complex</td>
<td>real</td>
</tr>
<tr>
<td>**CBAIEX (C, I)</td>
<td>C**I</td>
<td>(to integer)</td>
<td>complex</td>
</tr>
<tr>
<td>CCOS (C)</td>
<td>Complex cosine</td>
<td>complex</td>
<td>complex</td>
</tr>
<tr>
<td>CEXP (C)</td>
<td>Complex exponential</td>
<td>complex</td>
<td>complex</td>
</tr>
<tr>
<td>CLOG (C)</td>
<td>Complex log function</td>
<td>complex</td>
<td>complex</td>
</tr>
<tr>
<td>COS (X)</td>
<td>Cosine X radians</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>CSIN (C)</td>
<td>Complex sine</td>
<td>complex</td>
<td>complex</td>
</tr>
<tr>
<td>CSQRT (C)</td>
<td>Complex square root</td>
<td>complex</td>
<td>complex</td>
</tr>
<tr>
<td>DATAN (D)</td>
<td>Double arctangent</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DATAN2(D1, D2)</td>
<td>Double arctangent D1/D2</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>**DBADEX(D1, D2)</td>
<td>D1**D2</td>
<td>(to double)</td>
<td>double</td>
</tr>
<tr>
<td>**DBAEX (D, I)</td>
<td>D**I</td>
<td>(to integer)</td>
<td>double</td>
</tr>
<tr>
<td>**DBAREX(D, X)</td>
<td>D**X</td>
<td>(to real)</td>
<td>double</td>
</tr>
<tr>
<td>DCS (D)</td>
<td>Double cosine</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DEXP (D)</td>
<td>Double exponential function (e^D)</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DLOG (D)</td>
<td>Natural log of D (base e)</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DLOG10 (D)</td>
<td>Log to base 10 of D</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DMAX1</td>
<td>Maximum of n arguments</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DMIN1</td>
<td>Minimum of n arguments</td>
<td>double</td>
<td>double</td>
</tr>
</tbody>
</table>

* For detailed forms of the functions called, see AMAX/AMIN writeup.

** These routines must be called with a ** operator. They may not be used as functions.
**Fortran Library Functions (cont'd)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Parameter Type</th>
<th>Mode of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMOD(D₁,D₂)</td>
<td>D₁ modulo D₂</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>D₁ - (AINT(D₁/D₂)D₂)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSIGN(D₁,D₂)</td>
<td>Sign of D₂ times</td>
<td>D₁</td>
<td></td>
</tr>
<tr>
<td>DSIN(D)</td>
<td>Sine of double-precision argument</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>DSQRT(D)</td>
<td>Square root of double</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>EXP(X)</td>
<td>e to xth power (e^x)</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>**IBAIX(I₁,I₂)</td>
<td>I₁ ** I₂</td>
<td>integer</td>
<td>integer</td>
</tr>
<tr>
<td>IDINT(D)</td>
<td>Convert double-precision argument to integral floating point number (rounded)</td>
<td>double</td>
<td>real</td>
</tr>
<tr>
<td>RANF(X)</td>
<td>Random number generator. Results between 0 and 1, rectangular distribution.</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>**RBAIX(X,I)</td>
<td>X**I</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>**RBAIX(X₁,X₂)</td>
<td>X₁**X₂</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>SIN(X)</td>
<td>Sine X radians</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>SNGL(D)</td>
<td>Convert double-precision argument to real</td>
<td>double</td>
<td>real</td>
</tr>
<tr>
<td>SQRT(X)</td>
<td>Square root of X</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>TAN(X)</td>
<td>Tangent X radians</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>TANH(X)</td>
<td>Hyperbolic tangent X radians</td>
<td>real</td>
<td>real</td>
</tr>
<tr>
<td>TIMEF(1)</td>
<td>Elapsed CPU time in milliseconds</td>
<td>--</td>
<td>real</td>
</tr>
</tbody>
</table>

** These routines must be called with a ** operator. They may not be used as functions.
See ASIN writeup, page 212.

Timing 68 µs
Purpose: To calculate the natural logarithm of a floating point argument X.

Fortran Function: \( E = \text{ALOG}(X) \) (\( E = \text{LOGF}(X) \))

Entry Points: ALOG(LOGF)

Ascent Calling Sequence:
- SA1
- RJ
- BX6
- SA6

X
ALOG
XI
E

Normal Return: The result returned in XI is stored in E.

Error Message: ARG ZERO OR NEG

Storage: 548 locations

Accuracy: The error in the worst case is about \( \pm 10^{-12} \).

Timing: 36 \( \mu s \)

Method:
\[
x = f \cdot 2^N, \quad 0 < f < 1
\]
\[
t = \frac{f - \sqrt{f}}{f + \sqrt{f}}
\]
\[
\log_e x = N \cdot \log_e 2 + \left[ \log_e \sqrt{2} + 2t + \frac{2t^3}{3} + \frac{2t^5}{5} + \ldots + \frac{2t^{13}}{13} \right]
\]
ALOG10

Purpose
Calculate the logarithm to the base 10 of a floating point argument.

Fortran Function
A = ALOG10(X)

Entry Points
ALOG10

Ascent Calling Sequence
SA1 X
RJ ALOG10
BX6 XI
SA6 A

Normal Return
The Result returned in XI is stored in A.

Error Message
ARG ZERO OR NEG

Storage
558 locations

Accuracy
Approximately 10^-13 at worst

Timing
39 µs

Method *
\[ \log_{10}(x) = 0.43429448190325 \log_e(x) \]
**AMAX/AMIN**

**Purpose**
To calculate the maximum (or minimum) of *n* arguments.

**Entry Points**

<table>
<thead>
<tr>
<th>Entry Points</th>
<th>Entry Forms</th>
<th>Mode of Parameters</th>
<th>Mode of Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMAX0(I₁, I₂ ... Iₘ)</td>
<td>integer</td>
<td>real</td>
<td></td>
</tr>
<tr>
<td>(MAXOF(I₁, I₂ ... Iₘ))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMAX1(X₁, X₂ ... Xₘ)</td>
<td>real</td>
<td>real</td>
<td></td>
</tr>
<tr>
<td>(MAX1F(X₁, X₂ ... Xₘ))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MINO(I₁, I₂ ... Iₘ)</td>
<td>integer</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>(MINOF(I₁, I₂ ... Iₘ))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN1(X₁, X₂ ... Xₘ)</td>
<td>real</td>
<td>integer</td>
<td></td>
</tr>
<tr>
<td>(MIN1F(X₁, X₂ ... Xₘ))</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Ascent Calling Sequence**

For \( R = \text{AMAXO}(I,J,K) \)

<table>
<thead>
<tr>
<th>RJ</th>
<th>MAXOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>*+4</td>
</tr>
<tr>
<td>CØN</td>
<td>I</td>
</tr>
<tr>
<td>CØN</td>
<td>J</td>
</tr>
<tr>
<td>CØN</td>
<td>K</td>
</tr>
<tr>
<td>BX6</td>
<td>X₁</td>
</tr>
<tr>
<td>SA6</td>
<td>R</td>
</tr>
</tbody>
</table>
**AMAX/AMIN (cont'd)**

<table>
<thead>
<tr>
<th><strong>Normal Return</strong></th>
<th>The result returned in X1 is stored in R.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error Message</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>778 locations</td>
</tr>
<tr>
<td><strong>Method</strong>*</td>
<td>A loop is set up to determine the maximum (minimum) argument by successive subtractions.</td>
</tr>
</tbody>
</table>
ASIN/ACOS

Purpose
This routine calculates both ASIN and ACOS using multiple entry points.

Purpose
To compute the single-precision arcsine (arccosine) of a single-precision argument.

Fortran Function
B = ASIN(X) (B = ASINF(X))
B = ACOS(X) (B = ACOSF(X))

Entry Points
ASIN(ASINF)
ACOS(ACOSF)

Ascent Calling Sequence
SAI RJ BX6 SA6 X ASIN XI B

Normal Return
The result in the range $\frac{-\pi}{2}$ to $\frac{\pi}{2}$ is returned in XI and stored in B.

Error Message
ARG GT 1.0 OR INDEFINITE OR INFINITE ARG FROM LOC

Storage
1368 locations

Accuracy
1.E-12 in the worst case.

Timing
ASIN 64 μs
ACOS 68 μs

Method
The method used is a power series telescoped by the use of Chebyshev polynomials. The range of X is partitioned into intervals, and values of constants are assigned accordingly in the approximation equation. Near the origin for values of X less than 0.05, a Maclaurin series is used. The results are in quadrant I or quadrant IV. ACOS is computed as $\frac{\pi}{2} - ASIN(X)$. The result returns $0 < B < \pi$; that is, in the first or second quadrant.
Library Routines - Mathematical

ATAN

Purpose
To calculate the arctangent of X in radians.
\((-\pi/2 < \text{result} < \pi/2)\)

Fortran Function
\(A = \text{ATAN}(X) \quad (A = \text{ATANF}(X))\)

Entry Points
\(\text{ATAN(\text{ATANF})}\)

Ascent Calling Sequence
\(\text{SA1} \quad X \quad \text{RJ} \quad \text{ATAN} \quad \text{BX6} \quad \text{XI} \quad \text{SA6} \quad A\)

Normal Return
The result returned in XI is stored in A. The result is in radians, and is between \(-\pi/2\) and \(\pi/2\).

Error Message
INDEFINITE OR INFINITE ARG FROM LOC

Storage
758 locations

Accuracy
About \(1.0 \times 10^{-14}\)

Timing
37 μs

Method
Let \(A = \text{ATAN}(X)\), then \(-\pi/2 < A < \pi/2\)
Let \(P = \tan(\pi/16)\), \(T = \tan(3\pi/16)\)
\(\text{ATAN}(W) = \text{sign}(W) \times \text{ATAN}(V)\), \(V = \text{ABS}(W)\)
\(\text{ATAN}(V) = \text{ATAN}(R)+C\), \(R\) and C defined below

\[
\begin{align*}
0 \leq V < P & \quad R = V & \quad C = 0.0 \\
P \leq V < 2^{\frac{1}{2}}-1 & \quad R = (V-P)/(1+V\times P) & \quad C = \pi/16 \\
2^{\frac{1}{2}}-1 \leq V < 1 & \quad R = (V-T)/(1+V\times T) & \quad C = 3\pi/16 \\
1 \leq V < 2^{\frac{1}{2}}+1 & \quad R = (1-V\times T)/(V+T) & \quad C = 5\pi/16 \\
2^{\frac{1}{2}}+1 \leq V & \quad R = (1-V\times P)/(V+P) & \quad C = 7\pi/16 
\end{align*}
\]
ATAN (cont'd)

Method (cont'd)

\[
\text{ATAN}(R) = R - R \times Q \\
Z = R^2 \\
Q = \frac{n_0 + n_1 Z + n_2 Z^2 + n_3 Z^3}{d_0 + d_1 Z + d_2 Z^2 + d_3 Z^3}
\]
ATAN2

Purpose: To calculate the arctangent of Y/X in the range
\(-\pi < \text{result} < \pi\).

Fortran Function: \( B = \text{ATAN2}(Y, X) \)

Entry Point: ATAN2

Ascent Calling Sequence:
- \( RJ \)
- \( EQ \)
- \( CONS \)
- \( CON \)
- \( BX6 \)
- \( SA6 \)

Normal Return: The result returned in XI is stored in B. The result in radians is between \(-\pi\) and \(\pi\).

Error Message: INDEFINITE OR INFINITE ARG OR X = Y = 0

Storage: 1218 locations

Accuracy: About \(1. \times 10^{-14}\)

Timing: 59 \(\mu\)s (average)

Method: Let \( B = \text{ATAN2}(Y, X) \), then \( B \) is the argument of the complex number \( X = iY \) and \(-\pi \leq B \leq \pi\).

\[
B = \begin{cases} 
-sin(Y)\times\pi/2 & X = 0 \\
\text{ATAN}(Y/X) & X > 0 \\
\text{-ATAN}(Y/X) + \text{sign}(Y)\times\pi & X < 0 
\end{cases}
\]

Let \( P = \tan(\pi/16) \), \( T = \tan(3\pi/16) \), \( W = Y/X \)

\( \text{ATAN}(W) = \text{sign}(W)\times\text{ATAN}(V), \ V = \text{ABS}(W) \)

\( \text{ATAN}(V) = \text{ATAN}(R) + C, \ R \) and \( C \) defined below
ATAN2 (cont'd)

Method (cont'd)

\[
\begin{align*}
0 \leq V < P & \quad R = V \quad C = 0.0 \\
P \leq V < 2^{\frac{3}{2}} - 1 & \quad R = (V-P)/(1+V*P) \quad C = \pi/16 \\
2^{\frac{3}{2}} - 1 \leq V < 1 & \quad R = (V-T)/(1+V*T) \quad C = 3\pi/16 \\
1 \leq V < 2^{\frac{3}{2}} + 1 & \quad R = (1-V*T)/(V+T) \quad C = 5\pi/16 \\
2^{\frac{3}{2}} + 1 \leq V & \quad R = (1-V*P)/(V+P) \quad C = 7\pi/16 \\
\end{align*}
\]

\[
ATAN(R) = R - R*Q, \quad Z = R^2
\]

\[
Q = \frac{n_0 + n_1Z + n_2Z^2 + n_3Z^3}{d_0 + d_1Z + d_2Z^2 + d_3Z^3}
\]
CABS

Purpose
To calculate the magnitude of a complex number.

Fortran Function
X = CABS(Z)

Entry Point
CABS

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ CABS
BX6 XI
SA6 X

Normal Return
The result (a real number) returned in XI is stored in X.

Error Message
None

Storage
32 locations

Accuracy
The worst case is $10^{-12}$. Average is $10^{-14}$.

Timing
26 μs (average)

Method
The result approximates

$$|Z| = \left( X^2 + Y^2 \right)^{\frac{1}{2}}$$

where $Z = X + iY$
To compute $Z^I$ where $Z$ is complex and $I$ is integer. The real part is in $Z$; the imaginary part, in $Z+1$.

**Fortran Function**

$C = Z^I$ where $C$ is complex.

**Entry Point**

CBAIEX

**Ascent Calling Sequence**

- RJ
- EQ
- Z (X0N)
- I (X0N)
- X1 (LX6)
- X2 (BX7)
- C (SA6)
- C+1 (SA7)

**Normal Return**

The real part of the result is in X1; the imaginary part, in X2. The result is stored in C.

**Error Message**

ZERO TO ZERO OR NEG POWER

**Storage**

368 locations

**Timing**

- 31 μs for an exponent of 2
- 43 μs for an exponent of 10

**Method**

Successive complex multiplications are performed in a loop depending on the exponent $I$. If $Z = 0$, $C = 0$ if $I > 0$.

If $Z = 0$ and $I \leq 0$, an exit will be forced and the panel and an error message will be printed. If $Z \neq 0$ and $I < 0$, $C = 1/C$. 
Library Routines - Mathematical

**CC0S**

Purpose: To calculate the cosine of the complex argument Z.

**Fortran Function**:

\[ C = \text{CC0S}(Z) \]

**Entry Point**:

\text{CC0S}

**Externals**:

\text{C0SF}, \text{SINF}, \text{EXPF}

**Ascent Calling Sequence**:

- \text{SA1} \quad \text{Z}
- \text{SA2} \quad \text{Z}+1
- \text{RJ} \quad \text{CC0S}
- \text{BX6} \quad \text{X}1
- \text{LX7} \quad \text{X}2
- \text{SA6} \quad \text{C}
- \text{SA7} \quad \text{C}+1

**Normal Return**:

The real part is in X1; the imaginary part in X2. The complex result is stored in C.

**Error Message**:

INDEFINITE OR INFINITE ARG FROM LOC

**Storage**:

418 locations

**Accuracy**:

About 10^-13.

**Timing**:

122 \mu s

**Method**:

\[ \text{CC0S}(Z) = \text{CSIN}\left(\frac{\pi}{2} - Z\right) \]
CEXP

Purpose
Calculate the exponential of a complex argument \( Z \).

Fortran Function
\( C = \text{CEXP}(Z) \)

Entry Point
CEXP

Externals
SINF, C0SF, EXPF

Ascent Calling Sequence
- \( Z \)
- \( Z+1 \)
- \( \text{CEXP} \)
- \( \text{X1} \)
- \( B0+C \)
- \( \text{X2} \)
- \( A6+1B \)

Normal Return
Real part is returned in \( \text{X1} \); imaginary part, in \( \text{X2} \). The complex result is stored in \( C \).

Error Messages
None in CEXP. See EXP, SIN, C0S.

Storage
148 locations

Accuracy
About \( 10^{-13} \)

Timing
107 \( \mu \)s

Method*
For complex \( Z = x+iy \)
\[
e^Z = e^x \cos(y) + ie^x \sin(y)
\]
EXP, C0S, SIN are called to evaluate this.
CLOG

Purpose
Calculate the complex logarithm, to the base e, of a complex argument Z.

Fortran Function
C = CLOG(Z)

Entry Point
CLOG

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ CLOG
BX6 X1
LX7 X2
SA6 C
SA7 C+1

Normal Return
The real part is in X1; the imaginary part, in X2.
The complex result is stored in C.

Error Message
ARG(0.,0.) FROM LOC

Storage
1438 locations

Accuracy
About 10^{-14}

Timing
72 µs (average)

Method
\[ X + iY = CLOG(C+CI) \]
\[ X = \frac{1}{2} \log_e(C^2 + CI^2) \]
\[ Y = \text{ATAN2}(CI,C) \]
See SIN writeup, page 250.
CSIN

Purpose
Calculate the sine of a complex number.

Fortran Function
R = CSIN(Z)

Entry Point
CSIN

Externals
CØSF, SINF, EXPF

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ CSIN
BX6 X1
SA6 R
LX7 X2
SA7 R+1

Normal Return
The real part is in X1; imaginary part, in X2.
The complex result is stored in R.

Error Message
INFINITE OR INDEFINITE ARG FROM LOC

Storage
418 locations

Accuracy
About 10^-13

Timing
122 μs

Method *
Where z = x + iy

\[ \text{csin}(z) = \sin(x) \frac{e^y + e^{-y}}{2} + \cos(x) \frac{e^y - e^{-y}}{2} \]
### CSQRT

**Purpose**
Calculate the square root of a complex number $Z$.

**Fortran Function**

$$ X = \text{CSQRT}(Z) $$

**Entry Point**

CSQRT

<table>
<thead>
<tr>
<th>Ascent Calling Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>$Z$</td>
</tr>
<tr>
<td>SA2</td>
<td>$Z+1$</td>
</tr>
<tr>
<td>RJ</td>
<td>CSQRT</td>
</tr>
<tr>
<td>BX6</td>
<td>$X1$</td>
</tr>
<tr>
<td>SA6</td>
<td>$X$</td>
</tr>
<tr>
<td>LX7</td>
<td>$X2$</td>
</tr>
<tr>
<td>SA7</td>
<td>$X+1$</td>
</tr>
</tbody>
</table>

**Normal Return**

The real part is returned in $X1$; the imaginary, in $X2$.
The complex result is stored in $X$.

**Error Message**

None

**Storage**

508 locations

**Accuracy**

At worst, $10^{-13}$.

**Timing**

49 µs (average)

**Method**

For $Z = A + iB$

$$ X + iY = (A+iB)^{\frac{1}{2}} $$

$$ X = \sqrt{\left[(A^2 + B^2)^{\frac{1}{2}} + A\right] / 2} $$

$$ Y = \frac{B}{2X} $$

Results are returned between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. 
Library Routines - Mathematical

DATAN

Purpose
To compute the double-precision arctangent of a double-precision argument X. Results are in radians where

\[-\frac{\pi}{2} < Y < \frac{\pi}{2}.\]

Fortran Function
\[Y = \text{DATAN}(X)\]

Entry Point
\[\text{DATAN}\]

Ascent Calling Sequence
- \(X\)
- \(X+1\)
- \(\text{DATAN}\)
- \(X1\)
- \(X2\)
- \(Y\)
- \(A6+1\)

Normal Return
The most significant part is left in \(X1\); the least significant part, in \(X2\). The result is stored in \(Y\).

Error Message
INDEFINITE OR INFINITE ARGUMENT FROM LOC

Storage
176 locations

Accuracy
Average is \(10^{-28}\)

Timing
Approximately 136 \(\mu\)s

Method
DATAN is computed from a polynomial telescoped from the Taylor-Maclaurin power series of degree 39. The result is set to the argument where \(-1.0E-10 < \text{arg} < 1.0E-10\).
DATAN2

Purpose To compute in double-precision the angle in radians when the tangent is given in terms of coordinates X and Y, where X and Y are double-precision. The angle in radians is between $\pi$ and $\pi$ ($-\pi < \text{result} < \pi$).

Fortran Function

Fortran Function

Fortran Function

R = DATAN2(Y, X)

Entry Point DATAN2

Ascent Calling Sequence

Ascent Calling Sequence

Ascent Calling Sequence

RJ
EQ
C0N
C0N

DATAN2

DATAN2

DATAN2

DATAN2

Normal Return

Normal Return

Normal Return

The result between $-\pi$ and $\pi$ is in radians. The most significant part is left in X1, the least significant part in X2. The result is stored in R.

Error Message

Error Message

Error Message

X = Y = 0 or INDEFINITE OR INFINITE ARGUMENT FROM LOC

Storage

Storage

Storage

2118 locations

Accuracy

Accuracy

Accuracy

Relative error about $10^{-28}$

Timing

Timing

Timing

Approximately 130 $\mu$s

Method

Method

Method

DATAN2 establishes the correct quadrant based on the sign of the coordinates X and Y. It is computed from a polynomial telescoped from the Taylor-Maclaurin power series of degree 39.
DATAN2 (cont'd)

Method (cont'd)

If $X = 0$, $R = \text{sign}(Y) \times \frac{\pi}{2}$

$X > 0$, $R = \text{DATAN}(Y/X)$

$X < 0$, $R = \text{DATAN}(Y/X) + \text{sign}(Y) \times \pi$

For values of $Y/X$ less than $10^{-10}$, the result is set to the value of the argument.
Purpose
To calculate the D to the DD power, where D and DD are double-precision.

Fortran Function
D2 = D**DD

Entry Point
DBADEX

Ascent Calling Sequence
<table>
<thead>
<tr>
<th>RJ</th>
<th>DBADEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ</td>
<td>*+3</td>
</tr>
<tr>
<td>CON</td>
<td>D</td>
</tr>
<tr>
<td>CON</td>
<td>DD</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>LX7</td>
<td>X2</td>
</tr>
<tr>
<td>SA6</td>
<td>D2</td>
</tr>
<tr>
<td>SA7</td>
<td>D2+1</td>
</tr>
</tbody>
</table>

Normal Return
The double-precision result is stored in D2; the most significant part is returned in X1 and the least significant in X2.

Error Message
NEG BASE WITH REAL EXP FROM LOC

Storage
2218 locations

Accuracy
About 10^-25

Timing
237 μs
Purpose

To calculate $D^I$ ($D^I$) where the argument $D$ is double-precision and $I$ is integer.

Fortran Function

$DD = D^I$

Entry Point

DBAIEX

Ascent Calling Sequence

RJ  DBAIEX
EQ  *+3
CØN  D
CØN  I
BX6  X1
LX7  X2
SA6  DD
SA7  DD+1

Normal Return

The most significant part is returned in X1, the least significant part in X2. Double-precision results are stored in DD.

Error Message

None; an overflow will be returned from PM

Storage

348 locations

Accuracy

The error compared with successive double-precision multipliers is $10^{-26}$.

Timing

For $I = 2$, 33 µs
For $I = 12$, 49 µs
Time increases as $I$ increases.

Method

Successive double-precision multiplies are taken.
Purpose
To calculate the double-precision result of double base to real exponent.

Fortran Function
\[ D = DD^{**}R \]

Entry Point
DBAREX

Ascent Calling Sequence
- RJ
- EQ
- CON
- CN
- BX6
- LX7
- SA6
- SA7

Normal Return
The double-precision result is stored in D. The most significant part is returned in X1; the least significant part in X2.

Error Message
NEG BASE WITH REAL EXP FROM

Storage
2148 locations

Accuracy
About 10^{-26}

Timing
120 \mu s
<table>
<thead>
<tr>
<th>Library Routines - Mathematical</th>
</tr>
</thead>
</table>

#### DCOS

**Purpose**
Calculate double-precision cosine of a double-precision argument.

**Fortran Function**

\[ Y = \text{DCOS}(Z) \]

**Entry Point**

DCOS

**Ascent Calling Sequence**

- SA1 \( Z \)
- SA2 \( Z+1 \)
- RJ \( \text{DCOS} \)
- BX6 \( X1 \)
- LX7 \( X2 \)
- SA6 \( Y \)
- SA7 \( Y+1 \)

**Normal Return**

A double-precision result is stored in Y. The most significant part is returned in X1 and the least significant part in X2.

**Error Message**

\[ \text{ABS(ARG)} = \pi \times 2 \text{ TO } 94 \text{ FROM} \]

**Storage**

163, locations

**Accuracy**

About \( 10^{-28} \)

**Timing**

99 \( \mu s \)

**Method**

Similar to DSIN.
DEXP

Purpose: To compute the double-precision natural exponential of a double-precision argument D.

Fortran Function: 

Fortran Y = DEXP(D)

Entry Point: DEXP

Ascent Calling Sequence:

SA1 D
SA2 D+1
RJ DEXP
BX6 X1
LX7 X2
SA6 Y
SA7 A6+1

Normal Return: The most significant part is returned in X1; the least significant part, in X2. The double-precision result is stored in Y.

Error Message: ARG GREATER THAN 732. FROM LOC

Storage: 126 locations

Accuracy: Average is 10^{-29}. Tests performed by CDC for 5000 values of X uniformly distributed in the range |X| < Log_e(2)/2 found the maximum observed relative error was 3.6E-29. For 5000 values < 600., it was 8.0E-29.

Timing: Approximately 112 µs
DEXP (cont'd)

Let \( N = \left[ \frac{X}{\log_e(2.0)} + 0.5 \right] \), and \( R = R_1 + R_2 = X - N \log_e(2.0) \), \(|R| \leq \log_e(2.0)/2 \). \( R_1 \) and \( R_2 \) represent the more significant and less significant parts of \( R \).

\[
X e^r = 2 \times e^{R_1} \times e^{R_2}
\]

\( e^{R_1} \) is evaluated from a polynomial of degree 17. This polynomial was telescoped from a truncated Maclaurin power series of degree 20.

\[
e^{R_2} = 1 + R_2.
\]
DL0G

**Purpose**
To compute the double-precision natural logarithm of a double-precision argument D.

**Fortran Function**
Y = DL0G(X)

**Entry Point**
DL0G

<table>
<thead>
<tr>
<th>Ascent Calling Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>X</td>
</tr>
<tr>
<td>SA2</td>
<td>X+1</td>
</tr>
<tr>
<td>RJ</td>
<td>DL0G</td>
</tr>
<tr>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td>LX7</td>
<td>X2</td>
</tr>
<tr>
<td>SA6</td>
<td>Y</td>
</tr>
<tr>
<td>SA7</td>
<td>Y+1</td>
</tr>
</tbody>
</table>

**Normal Return**
The most significant part is in X1; the least, in X2. The double-precision result is stored in Y.

**Error Message**
ARG ZERO OR NEGATIVE FROM LOC

**Storage**
1608 locations

**Accuracy**
Average is $10^{-29}$. In tests done by CDC for 2000 values of X uniformly distributed in the range $0.5 \leq X \leq 2$, the maximum observed absolute error was 2.4E-29.

**Timing**
Approximately 133 µs

**Method**
\[ x = 2^W, \quad 2^{-\frac{1}{2}} < W < 2^{\frac{1}{2}} \]

\[ \log_e(x) = K \log_e 2 + \log_e W \]
Method
(cont'd)

Log\textsubscript{e} W is approximated by the equation

\[ \log_{e} W = C_1 t + C_3 t^3 + C_5 t^5 + C_7 t^7 \]

where \( t = (W-1)/(W+1) \)

If \( X \leq 0 \), an error exit is taken and a diagnostic printed.
DLG10

Purpose
To compute log to the base 10 of a double-precision argument.

Fortran Function
DD = DLG10(D)

Entry Point
DLG10

Ascent Calling Sequence
SA1 D
SA2 D+1
RJ DLG10
BX6 XI
LX7 X2
SA6 DD
SA7 DD+1

Normal Return
The most significant part is returned in X1, the least significant part in X2. Double-precision results are stored in DD.

Error Message
ARG ZERO OR NEG FROM LOC

Storage
1618 locations

Accuracy
About $10^{-28}$

Timing
About 138 μs

Method
$log_{10}(x) = \log_{10}(e) \times \log_{e}(x)$
Purpose

To calculate the maximum (minimum) of n arguments.

Fortran Function

\[ D_1 = \text{DMAX1}(D_1, D_2, \ldots, D_n) \]
\[ DD = \text{DMIN1}(D_1, D_2, \ldots, D_n) \]

Entry Points

DMAX1, DMIN1

Ascent Calling Sequence

For \( D = \text{DMAX1}(D_1, D_2, D_3) \):

- \( RJ \)  \( \text{DMAX1} \)
- \( EQ \)  \( \*44 \)
- \( CON \)  \( D_1 \)
- \( CON \)  \( D_2 \)
- \( CON \)  \( D_3 \)
- \( BX6 \)  \( X_1 \)
- \( BX7 \)  \( X_2 \)
- \( SA6 \)  \( D \)
- \( SA7 \)  \( D+1 \)

Normal Return

The double-precision result is returned in \( X_1 \) and \( X_2 \).

Error Message

None

Storage

248 locations

Accuracy

It is exact.

Method

The most significant portion is checked for a maximum (minimum). If the difference is zero between two arguments, then the least significant portion is checked.
### DMOD

**Purpose**
To calculate the value of the double-precision argument D1 modulo D2.

**Fortran Function**
\[ D = \text{DMOD}(D1, D2) \]

**Entry Point**
DMOD

**Ascent Calling Sequence**
- RJ
- EQ
- CON
- CON
- BX6
- BX7
- SA6
- SA7

**Normal Return**
The most significant part is returned in X1, the least significant part in X2. Double-precision results are stored in D.

**Error Message**
INDEF OR INF ARG, X = 0, OR INTEGRAL PART TOO LARGE

**Storage**
42 locations

**Accuracy**
The formula is exact.

**Timing**
50 \( \mu s \)

**Method**
\[ D = D1 - \lfloor (D1/D2) \rfloor D2 \]
where \( \lfloor D1/D2 \rfloor = \text{integral part} \)
**DSIGN**

**Purpose**
To transfer the sign of argument D2 to the absolute value of argument D1.

**Fortran Function**

\[ D = \text{DSIGN}(D1,D2) \]

**Entry Point**
DSIGN

<table>
<thead>
<tr>
<th>Ascent Calling Sequence</th>
<th>RJ</th>
<th>EQ</th>
<th>C0N</th>
<th>C0N</th>
<th>BX6</th>
<th>BX7</th>
<th>SA6</th>
<th>SA7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DSIGN</td>
<td>( \ast +3 )</td>
<td>D1</td>
<td>D2</td>
<td>X1</td>
<td>X2</td>
<td>D</td>
<td>D+1</td>
</tr>
</tbody>
</table>

**Normal Return**
The most significant part is returned in X1, the least significant result is left in X2. The double-precision result is stored in D.

**Error Message**
None

**Storage**
138 locations

**Accuracy**
It is exact.

**Timing**
17 \( \mu \)s

**Method**
The sign of the second argument D2 is assigned to both the least and most significant parts of D1 without regard to the sign of D1.
DSIN

Purpose
To calculate the double-precision sine of a double-precision argument.

Fortran Function
Y = DSIN(Z)

Entry Point
DSIN

Ascent Calling Sequence
SA1 Z
SA2 Z+1
RJ DSIN
BX6 X1
LX7 X2
SA6 Y
SA7 Y+1

Normal Return
The most significant part is stored in X1, the least significant part in X2. The double-precision result is stored in Y.

Error Message
ABS(ARG) = PI X 2 TO 94 FROM

Storage
1638 locations

Accuracy
About 10^-28

Timing
91 μs

Method
N = \frac{x}{\pi/2} + .5
R = x - \frac{N\pi}{2} , then |R| ≤ \frac{\pi}{4}
K = N(MOD4)

Then
\sin(X) = \sin(R + \frac{N\pi}{2})
= \sin(R)\cos\left(\frac{K\pi}{2}\right) + \cos(R)\sin\left(\frac{K\pi}{2}\right)

Polynomials were telescoped using truncated Taylor-Maclaurin power series of degree 25 and 24.
**DSQRT**

**Purpose**
To calculate the double-precision square root of a double-precision argument.

**Fortran Function**
DD = DSQRT(D)

**Entry Point**
DSQRT

<table>
<thead>
<tr>
<th>Ascent Calling Sequence</th>
<th>SA1</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SA2</td>
<td>D+1</td>
</tr>
<tr>
<td></td>
<td>RJ</td>
<td>DSQRT</td>
</tr>
<tr>
<td></td>
<td>BX6</td>
<td>X1</td>
</tr>
<tr>
<td></td>
<td>LX7</td>
<td>X2</td>
</tr>
<tr>
<td></td>
<td>SA6</td>
<td>DD</td>
</tr>
<tr>
<td></td>
<td>SA7</td>
<td>DD+1</td>
</tr>
</tbody>
</table>

**Normal Return**
The most significant part is stored in X1, the least significant part in X2. Double-precision results are stored in DD.

**Error Message**
NEGATIVE ARGUMENT FROM

**Storage**
438 locations

**Accuracy**
$10^{-27}$

**Timing**
Average 42 μs

**Method**
Two Newton's iterations are done in single-precision, two more in double-precision.
EXP

Purpose
To calculate the exponential of a floating point argument.

Fortran Function
Y = EXP(X)

Entry Points
EXP(EXPF)

Ascent Calling Sequence
SA1 X
RJ EXP
BX6 XI
SA6 Y

Normal Return
The results returned in XI are stored in Y.

Error Message
ARG IN EXP GT 741 FROM LOC

Storage
458 locations

Accuracy
Approximately 10^{-13}

Timing
30 μs

Method*
Let \( N = [X \log_2(e) + .5] \), and \( R = X - N \log_2(2.0) \), then

\[
|R| \leq \log_2(2.0)/2
\]

\[
Z = R^2
\]

\[
B = Z^2 + 420.0Z + 15120.0
\]

\[
T = 28.0Z + 2520.0
\]
EXP (cont'd)

Method (cont'd)

\[ Q = \frac{R \cdot (R \cdot B - Z \cdot T)}{2 \cdot B + Z \cdot T - R \cdot B} \]

\[ e^R = 1. + R + Q \]

\[ e^X = 2^N \cdot e^R \]
IBAIEX

Purpose
To calculate the integer result of integer base to integer exponent ($I^J$).

Fortran Function
$K = J**I$

Entry Point
IBAIEX

Ascent Calling Sequence
- RJ
- EQ
- CON
- CON
- BX6
- SA6

Normal Return
The results returned in X1 are stored in K.

Error Message
0 BASE TO 0 EXPONENT OR RESULT EXCEEDS 2 TO 48 FROM LOC

Storage
308 locations

Accuracy
The method is exact.

Timing
The time ranges from 26 μs to 81 μs depending on the value of the exponent. For an exponent of 3, 31 μs; for an exponent of 12, 50 μs.

Method*
Form $I^J$ by successive multiplications of the base.
- If $J = 0$, $I \neq 0$, set to 1.
- If $J = 0$, $I = 0$, error exit.
- If $J \neq 0$, $I = 0$, set to 0.

Note: If $J$, the exponent, is a constant and equals zero, the compiler will generate code which sets the result to 1. No error exit will occur, as there is no entry to IBAIEX in this case.
IDINT

Purpose
Convert double-precision argument to a floating point integral argument.

Fortran Function
A = IDINT(D)

Entry Point
IDINT

Ascent Calling Sequence
SA1
SA2
D
D+1
RJ
IDINT
BX6
XI
SA6
A

Normal Return
The result returned in XI is stored in A.

Error Message
None

Storage
258 locations

Accuracy
The method is exact.

Timing
21 µs

Method
The argument is shifted to integer and the least significant part of the integer is added to get an exact result as a full word integer, which is packed and normalized.

Example:
5.4316999999999999D+13 converts to 5431700000000.00

A floating point integral number is returned.
RANF

Purpose
To generate a uniformly distributed sequence of random numbers between 0 and 1 in floating point format, use RANF as a function repeatedly for the set. To change the generating number, call RANSET as a subroutine.

Fortran Function
\[ X = \text{RANF}(A) \]

Entry Points
RANF, RANSET

Ascent Calling Sequence
- RJ
- EX6
- SA6

Normal Return
The results returned in X1 are stored in X.

Error Message
None

Storage
138 locations

Accuracy
The \( \chi^2 \) test on 10,000 numbers was satisfactory.

Timing
12 \( \mu \)s

Method*
Number_{i+1} = Number_i \times K \quad \text{where Number and K are packed as floating integers.}

\[ X = \text{Number}_{i+1} \mod 2^{48} \]
\[ K = 2^{24} - 3 \]

The argument A is a dummy and is not used by the function.

Use a floating point argument to avoid extra pack and normalize instructions added by the compiler (where RANSET has not previously been called, the generative number is set to 9).
RANF (cont'd)

Fortran

CALL RANSET(I)

The generative number \( \text{NUMBER}_0 \) is reset to the integer specified by the argument \( I \), and the repeated use of \( X = \text{RANF}(1.0) \) following a \text{CALL RANSET}(I) generates a set of random numbers based on \( I \). To repeat a set of random numbers, this generative number must be reset to the generative number of the original set. \( I \) should be an odd integer between 1 and \( 2^{20} \). An octal constant may also be used as the argument.

Ascent Calling

\[ \text{RJ} \quad \text{RANSET} \]
\[ \text{EQ} \quad \ast+2 \]
\[ \\
\text{CON} \quad I \]

Fortran

CALL RANGET(I)

The generative number currently in use by RANF is returned in \( I \). This number may be reentered by a \text{CALL} to \text{RANSET} to repeat a set of random numbers.

Ascent Calling

\[ \text{RJ} \quad \text{RANGET} \]
\[ \text{EQ} \quad \ast+2 \]
\[ \\
\text{CON} \quad I \]
RBAIEX

Purpose
To evaluate $X^I$, floating base to integer exponent.

Fortran Function
$Y = X**I$

Entry Point
RBAIEX

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ</td>
<td>RBAIEX</td>
</tr>
<tr>
<td>EQ</td>
<td>$I + 3$</td>
</tr>
<tr>
<td>CON</td>
<td>X</td>
</tr>
<tr>
<td>CON</td>
<td>I</td>
</tr>
<tr>
<td>BX6</td>
<td>XI</td>
</tr>
<tr>
<td>SA6</td>
<td>Y</td>
</tr>
</tbody>
</table>

Normal Return
The results returned in XI are stored in Y.

Error Message
ZERO BASE TO ZERO OR NEG EXPONENT FROM LOC

Storage
238 location

Accuracy
The method is exact. $N = \log_2 |I|$ is an upper bound for the number of binary bits lost due to rounding error.

Timing
About 23 $\mu$s

$$ (17 + 2.4 \times \log_2 |I|) \mu s \quad (+ 1.7 \mu s \text{ if } I \text{ neg}) $$

Method

$X^I = \sum_{i=0}^{n} a_i 2^i$

so

$$ X^I = \prod_{i=0}^{n} X a_i 2^i $$

Form $X^i$ $i = 1,n$ by successive squarings and multiply into the result if $a_i = 1$ in the binary configuration of $I$. 
RBAREX

Purpose
To calculate $X^Y$ where the arguments $X$ and $Y$ are real.

Fortran Function
$C = X^{**}Y$

Entry Point
RBAREX

Ascent Calling Sequence

| RJ | RBAREX |
| EQ | *+3    |
| CON | X     |
| CON | Y     |
| BX6 | X1    |
| SA6 | C     |

Normal Return
The result returned in X1 is stored in C.

Error Message
EXPONENT OVERFLOW OR NEG BASE IN X TO Y FROM LOC

Storage
1408 locations

Accuracy
About $10^{-12}$

Timing
89 $\mu$s

Method
$x^y = e^{(y \log x)}$
SIN/C0S

Purpose
Evaluate the sine or cosine of a floating point argument X, where X is in radians.

Fortran Function
Y = SIN(X)

Entry Points
SIN (SINF)
C0S (C0SF)

Ascent Calling Sequence
SA1 X
RJ SIN
EX6 XI
SA6 Y

Normal Return
The result returned in XI is stored in Y.

Error Message
ARG TOO LARGE, LOC

Storage
708 locations

Accuracy
Approximately 10^-14

Timing
33 μs

Method
Argument is first reduced to X ≤ π/6

then \( \sin(X) \approx X \sum_{N=0}^{5} C_i Z^N \)

where \( Z = X^2 \) and \( C_i = \frac{(-1)^i}{(2i+1)!} \)

C0S(X) is computed as SIN(\( \pi/2 - X \))
### SNGL

#### Purpose
To convert the double-precision argument to a single-precision result which has been rounded.

#### Fortran Function
\[ R = \text{SNGL}(D) \]

#### Entry Point
\text{SNGL}

<table>
<thead>
<tr>
<th>Ascent Calling Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>D</td>
</tr>
<tr>
<td>SA2</td>
<td>D+1</td>
</tr>
<tr>
<td>RJ</td>
<td>SNGL</td>
</tr>
<tr>
<td>BX6</td>
<td>XI</td>
</tr>
<tr>
<td>SA6</td>
<td>R</td>
</tr>
</tbody>
</table>

#### Normal Return
The real part is returned in XI and stored in R.

#### Error Message
None

#### Storage
78 locations

#### Accuracy
The method is exact.

#### Timing
About 10 \( \mu s \)

#### Method*
The most significant and least significant parts of D are added using the round floating add to effect a round. A zero test on the argument (D) sets XI to zero if D = 0.
SQRT

Purpose: Evaluate the square root of a real number.

Fortran Function: \( Y = \text{SQRT}(X) \)

Entry Points: SQRT (SQRTF)

Ascent Calling Sequence:
- SA1: \( X \)
- RJ: SQRT
- BX6: XI
- SA6: Y

Normal Return: The result returned in XI is stored in Y.

Error Message: NEGATIVE ARGUMENT IN SQRT FROM LOC

Storage: 348 locations

Accuracy: Average relative error is \( 10^{-13} \)

Timing: 23 \( \mu \)s

Method: The results of two Newton's iterations with a modified initial guess for \( \sqrt{X} \) are used.
TAN

Purpose
To calculate the tangent of X in radians.

Fortran Function
Y = TAN(X)

Entry Points
TAN (TANF)

Ascent Calling Sequence
SA1 XX
RJ TAN
BX6 X1
SA6 Y

Normal Return
The result returned in X1 is stored in Y.

Error Message
None

Storage
718 locations

Accuracy
Approximately $10^{-14}$. In the worst case, near $90^\circ$, it is about $10^{-10}$.

Timing
36 µs

Method
The routine subtracts a multiple of $2\pi$ to reduce the range.

- If $X > \pi$ \( \tan(X) = -\tan(2\pi - X) \).
- If $X > \frac{\pi}{2}$ \( \tan(X) = -\tan(\pi - X) \).
- If $X > \frac{\pi}{4}$ \( \tan(X) = -\cot(X - \frac{\pi}{2}) \).
### TANH

**Purpose**  
To calculate the hyperbolic tangent of X.

**Fortran Function**  
\( Y = \text{TANH}(X) \)

**Entry Points**  
\( \text{TANH (TANHF)} \)

**Ascent Calling Sequence**  
- \( \text{SA1} \quad X \)
- \( \text{RJ} \quad \text{TANH} \)
- \( \text{BX6} \quad X_1 \)
- \( \text{SA6} \quad Y \)

**Normal Return**  
The result returned in \( X_1 \) is stored in \( Y \).

**Error Message**  
None

**Storage**  
608 locations

**Accuracy**  
About \( 10^{-12} \). As the argument approaches zero between \(-10^{-4} \) and \( 10^{-4} \), the error increases rapidly to \( 10^{-8} \) and worse.

**Timing**  
52 μs

**Method**  
\( \text{TANH}(0) = 0 \)

\[
\text{TANH}(X) = \frac{e^{2X} - 1}{e^{2X} + 1} \quad \text{for} \quad |X| < 30.
\]

\[
\text{TANH}(X) = \pm 1. \quad \text{for} \quad |X| \geq 30.
\]
TIMEF

Purpose To obtain the current reading of the clock in milliseconds.

Fortran Function

\[ Z = \text{TIMEF}(1.0) \]

Entry Point TIMEF

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>SA1</th>
<th>RJ</th>
<th>BX6</th>
<th>SA6</th>
<th>X (where X = some number such as 1.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RJ</td>
<td>TIMEF</td>
<td>XI</td>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>

Normal Return The result returned in XI is stored in Z.

Error Message None

Storage 3 locations

Method* Cell 4 containing the clock counter is read, the value is packed and normalized, and the result is returned in XI. The argument 1. is a dummy variable. Cell 4 contains the total milliseconds of CPU time accumulated by the program at any given time.

Note: Even though the argument is a dummy, it should be floating point. Otherwise, the compiler will pack and normalize it.
UTILITY LIBRARY SUBROUTINES

GENERAL DESCRIPTION
A number of miscellaneous subroutines are also included in the library. Since they are of such varied character, no attempt is made to classify them. Instead, each routine is described briefly as to its use.

A subroutine communicates with the program which calls it by means of parameters in the list defining the "Call," or by means of Common. The formal parameters may be variables or arrays and are a "call by name"; that is, addresses (rather than values) of variables or arrays are sent to the subroutine.

A call would be compiled as follows:

```
CALL RØUT(A,B)
```

```
RJ  RØUT
EQ  *+3
CØN A
CØN B
```

The addresses of A and B are stored in locations following the return jump to the subroutine. Answers are returned by storing results in the names in the parameter list (or Common). A table of addresses for up to 64 parameters is generated by the compiler.
The following is a list of the routines described in this section. There are on the following pages in alphabetic order.

- ACGOER
- BACKSP (entries: IFENDF, REWINM)
- BUFFEI (entries: BUFFEO, IOCHEK, LENGTHF)
- BUFRD (entries: BUFINT, BUFWT, BUFRD, BUFCL)
- DEBUG
- DUMP
- ENDFIL
- EXIT (entries: END, STOP)
- GBYTE (entries: GBYTES, SBYTE, SBYTES)
- INPUTB (entry: OUTPTB)
- INPUTC
- INPUTS
- IOPROC (entries: RDTAPE, WRTAPE, IOWAIT, BSTAPE)
- KODER
- KRAKER
- OUTPTC
- OUTPTS
- OVERLAY
- PAUSE
- PDUMP
- RANRD (entries: RANINT, RANWT)
- RPTIN (entries: RPTIN, RPTOUT) (see section 600 for description)
- SAVEF
- SORT
- SSW (entry: SLT)
- TAPECY (see section 600 for description)
ACG0ER

Purpose To output an error message for an incorrectly computed GO TO statement.

Error Message ERROR IN COMPUTED GO TO, LOC. _____

Storage 5 locations

Method A branch to this routine is compiled in Fortran programs using a computed GO TO statement. The branch to SYSERR will be taken if the index is outside the range of the branches. The routine sets flags for the error message and jumps to cell 7.
BACKSP

Entry Points BACKSP, IFENDF, REWINM

Storage 778 locations

Entry BACKSP This entry backspaces one Fortran record on a tape. The unit number is contained in XI.

Fortran BACKSPACE i

Ascent Calling Sequence SA1 I
RJ BACKSP

Error Message POSITIONING ERROR. This occurs when trying to backspace a buffered record when the program is anticipating a Fortran record.

Entry IFENDF This entry checks for an end-of-file on unit i in the previous I/O statement. The branch is to n₁ for an end of file, branching to n₂ if none is encountered.

Fortran IF(EOF, i)n₁, n₂

Ascent Calling Sequence SA1 I
RJ IFENDF
ZR XI, statement #n₂
EQ B0, B0, statement #n₁

Entry REWINM This entry rewinds the tape to loadpoint.

Fortran REWIND i

Ascent Calling Sequence SA1 I
RJ REWINM
BUFFEI

Entry Points
BUFFEI, BUFFE0, I0CHEK, LENGTHF

Error Messages
UNASSIGNED UNIT
TOO MANY UNITS
LENGTHF CALL BEFORE I/O CALL
IF UNIT CALL BEFORE I/O CALL

Storage
2338 locations

Entry BUFFEI
This routine initiates a read of a physical tape record. The length is given in the call, and one of two modes, BCD or binary (even or odd parity), may be specified. A wait (RECALL) is not issued with the call in order that control may be returned to the caller, thus allowing programmer designed buffering.

Fortran
BUFFER IN (i,p)(A(l),A(100))

Ascent Calling Sequence
For BUFFER IN (I,1)(A(1),A(100)):
SB2 I
SB1 1
RJ BUFFEI
SB7 A
SB2 0
RJ BUFFEI
SB7 A+144B
SB2 0
RJ BUFFEI

The tape unit number or the address is stored in B2, the parity code in B1, after which a RJ to BUFFEI is made. The starting address of the block is stored in B7 with another RJ. Finally, the last word address is stored in B7 with the last call to BUFFEI. B2 is set to zero after the first call.
BUFFEI (cont'd)

**Entry BUFFEO**

See entry BUFFEI. This routine buffers out data.

**Fortran**

```
BUFFER OUT (i,p)(A(1),A(100))
```

**Ascent Calling Sequence**

For `BUFFER OUT (I,1)(A(1),A(100))`:

- `SB2`: I
- `SB1`: 1
- `RJ`: BUFFEO
- `SB7`: A
- `SB2`: 0
- `RJ`: BUFFEO
- `SB7`: A+144B
- `SB2`: 0
- `RJ`: BUFFEO

**Entry IÔCHEK**

This routine completes the call made with a previous `BUFFER OUT` or `BUFFER IN` call. A RECALL is issued if the previous I/O operation is incomplete, and then a status is returned. A status is returned immediately if the previous operation is complete.

**Fortran**

```
IF(UNIT,I)n_1,n_2,n_3,n_4
```

**Ascent Calling Sequence**

- `SA1`: I
- `RJ`: IÔCHEK
- `NG`: X1,n_1 (not ready)
- `ZR`: X1,n_2 (ready)
- `SX1`: X1-1
- `ZR`: X1,n_3 (end of file)
- `EQ`: n_4 (tape error)
BUFFEI (cont'd)

Entry IOCHEK (cont'd)

Ascent Calling Sequence (cont'd)
The unit I is stored in X1 and a RJ is taken to IOCHEK. The return flag is in X1.

X1 = 0  
X1 = negative  
X1 = 1  
X1 = 2

if unit is ready
if not ready
if EOF
if tape error

Branches are taken on the value of X1 to statements n1, n2, n3, and n4. A RJ to this entry checks the status of the unit in buffered I/O.

Entry LENGTHF

This routine returns the number of words transferred in a previous BUFFER IN or BUFFER OUT call. A RECALL is issued if the previous I/O operation is incomplete and then the length returned. The length is returned immediately if the previous I/O operation is complete.

Fortran

K = LENGTHF(I)

Ascent Calling Sequence

SA1            I
RJ            LENGHTF
BX6            X1
SA6            K

The address of the unit number is stored in X1 and a RJ to LENGTHF is taken. The answer is returned in X1. This entry returns the number of words in a buffered I/O operation on Unit I.
Purpose These programs enable a user to design the data movement schemes for special input-output problems whereby concurrent input and output can be maintained during a compute phase.

Description This set of programs provides an internal data processing system. Core is provided, according to the user's needs, as intermediate storage between the central memory and the peripheral drum memory. This storage area is called the Buffer. There are two Buffers: one for input data and one for output data. Each of these Buffers is divided in half so that the data may be passed through the alternate halves in a synchronized flip-flop technique.

In order to use these programs for the purpose of maintaining concurrent input or output during a compute phase, it is necessary to have the type of problem where computations can proceed on one set of data while another set is being loaded or unloaded. The computations must be of such length for each set that all I/O can be reasonably completed. Both the input and the output phases will be described. It is assumed initialization has been done.
Input Phase

Assume fifty logical records have been placed on drum -A- during initialization phase. This drum unit is rewound and the procedure starts. A call is made for Record 1. Record 1 is moved into the first half of the input buffer and a wait condition exists until it is in. Record 2 starts into the second half of the input buffer, but this time no wait condition is activated. Instead, while the second record is entering, the first record is moved into the designated array of the call and control returned to the caller. The caller now uses the new data for his problem. When the program calls for the second record, checks are made to make sure all the data for the second record has been moved into the second half of the input buffer. If the compute time was long enough, all of the data will be in; if not, then a wait condition will be activated until it is. The second record is moved into the designated array of the call, and the third record is started into the first half of the input buffer. Control is then returned to the caller. This procedure continues for the fifty records. This flip-flop method synchronizes the moving of data into alternate halves of the buffer and provides concurrent compute capability, if the compute time is sufficiently long.
Output Phase

Upon completion of the calculation which uses the first record, the output phase is started. Assume drum unit -B- has been rewound. A call is made to put the results of the first calculation onto the drum. The record sizes are the same in both input and output phases. Result Set 1 is put into the first half of the output buffer from the array designated in the call. Control is immediately returned to the caller. The next call is subsequently made to move Result Set 2 onto the drum. At this time the first half from the previous call starts to move to the drum, but no wait condition is activated. The second half of the output buffer is loaded from the array designated in the call and control returned to the caller. The next call is made to move Result Set 3 on to the drum. At this time checks are made to be sure all of Result Set 1 has been moved out of the first half of the buffer. If the compute time since the last call was long enough, all of the data will be moved out; if not, then a wait condition will be activated until it is. The second result set is started to the drum next. The first half of the output buffer is loaded from the array designated in the call and control is returned to the caller. This procedure continues until fifty result sets have been sent via the call.
Rewind and Exchange Units

After the fifty records have been read and fifty result sets calculated, the unit designations are rewound and exchanged. Unit -B- becomes the input unit, and unit -A- becomes the output unit. The complete cycle starts again using the results calculated in the last cycle for input.
Library Routines - Utility

BUFRD (cont'd)

Entry Points
BUFIN, BUFWT, BUFRD, BUFCL

Storage
3448 locations

Entry BUFIN

Fortran
BUFIN(EXTY,ENTRY,LENGTH,UNIT1,UNIT2)

Ascent Calling
Sequence
RJ          BUFINT
EQ          *+6
CON         EXYT
CON         ENTRY
CON         LENGTH
CON         UNIT1
CON         UNIT2

Description
The BUFIN subroutine is the initializing routine and must be called first. Two arrays, in addition to the data arrays, are required for buffering, and two logical unit numbers must be assigned to the drum. If the data array is dimensioned for 1000, then EXYT and ENTRY must be dimensioned for 2000 in the calling program.

Example

The parameters for BUFIN are defined below:

EXYT An array which is twice the length of the data array which is to be moved to the drum memory.
Library Routines - Utility

BUFRD (cont'd)

Entry BUFINT (cont'd)
ENTRY Another array (equal in length to the EXIT array) which processes data from the drum memory.
LENGTH Number of elements in EXYT. (EXYT and ENTRY must have the same dimensions in the calling program.)
UNIT1 A logical unit number to be used for the data passing through the EXYT array to the drum. This number must appear on an ASSIGN card, e.g., *ASSIGN,DRUM=i.
UNIT2 A second logical unit number to be used for the data passing through the ENTRY array from the drum. This number must also appear on an ASSIGN card, e.g., *ASSIGN,DRUM=i+1.

The BUFINT routine is called only once.

Entry BUFWT

Fortran

BUFWT(ARRAY,NERROR,NUNIT)

Ascent Calling Sequence

RJ  BUFWT
EQ  *+4
CÒN  ARRAY
CÒN  NERROR
CÒN  NUNIT

Description

The BUFWT subroutine is called when the data array called ARRAY is to be sent to the drum memory. The BUFWT routine moves the data from the ARRAY into the EXYT array when BUFWT is called and immediately returns to the caller. BUFWT then moves the data to the drum memory. Alternate halves of the EXYT routine are used to move data to the drum so that after two times through the loop, data is entering one half the EXYT array from ARRAY,
BUFRD (cont'd)

Entry BUFWT (cont')

Description (cont'd)

and at the same time data is moving to the drum from the other half of the EXYT array.

The arguments are defined as follows:

- **ARRAY**  The array containing data to be buffered.
- **NERROR**: 
  - **NERROR = 0**  No write error.
  - **NERROR ≠ 0**  An error occurred during the previous attempt to move data to or from the drum. This is an unrecoverable error, and the user should terminate.
- **NUNIT**  The unit number on which the error occurred. This number should be printed out for debugging purposes if NERROR ≠ 0.

BUFRD (ARRAY, NERROR, NUNIT)

Description

The BUFRD subroutine is called when the data array is to be loaded from the drum memory. The BUFRD routine moves the data from alternate halves of the ENTRY array into ARRAY and returns to the caller. BUFRD then starts the next read from the drum memory in order to have data ready for the next call. See BUFWT for definition of arguments.
**Library Routines - Utility**

**BUFRD (cont'd)**

**Entry BUFCL**

**Fortran**

BUFCL(N)

**Ascent Calling Sequence**

- RJ
- EQ
- CON

**Description**

The BUFCL subroutine rewinds the units assigned to the EXYT and ENTRY arrays. BUFCL must be called when repositioning is required and is used in a manner similar to the ordinary rewind instruction in tape usage.

- If $N = 0$, the unit numbers are exchanged in order that the input unit will be used for output and the output unit will be used for input.

- If $N = 1$, the unit numbers are not exchanged, but the rewind takes place.

**Sample Program**

The following example is given to illustrate the use of these routines. Both the N1 and N2 arrays are used in this program to make it possible to check that data is moving in and out correctly. Only one of these arrays, in addition to the exit and entry array, would be needed by BUFRD to do true buffering. The array N1 is initialized and moved to the drum before the buffering loop starts.
PROGRAM BUF

COMMON N1(4000),N2(4000),NX(8000),NN(8000)
NUL=9
NU2=10

LOAD N1 WITH NO.S 1-4000
NX=0
NXA=1
NMAX=4000
DO 20 I=1,NMAX
NI(I)=NX
NX=NX+NXA
CONTINUE

FORMAT (1HI)
FORMAT (20(2X,I4))

INITIZE ARRAYS NX AND NN FOR BUFFERING
CALL BUFIN (NX,NN,8000+NU1+NU2)
NER=0
NUNI=0

BUILD 1ST WRITE BUFFER
DO 25 I=1,10
CALL BUFWT (N1,NER,NUNI)
CONTINUE
REWIND AND EXCHANGE UNITS
CALL BUFCL(0)

KP=0
GO TO 70
1=1,60
DO 60 J=1,10
N2 NOW READS CONTENTS OF N1 BECAUSE UNITS ARE EXCHANGED
CALL BUFRD (N2,NER,NUNI)

CONTINUE

PRINT STANDARD ARRAY FIRST TIME THROUGH
IF (KP) 301,300

DO FORTRAN CHECK FOR CORRECT READ AND SET FLAG COUNTERS FOR TEST
NORMAL AT THIS POINT A GREAT DEAL OF CPU CODE MUST APPEAR IN ORDER FOR THE I/O CALLS TO BE SPACED FAR ENOUGH IN TIME TO GET TRUE BUFFERING
KE =0
KDI =0
DO 42 K=1,4000
IF (NI(K)=N2(K)) 33,35,33
KE =KE +1
GO TO 42
35 KE =KE +1
42 CONTINUE
DEFINE A NEW NI ARRAY AND ERROR CHECK CO

KP=I
301 CONTINUE

301 CONTINUE

CONTINUE CALL BUFWT (Ni, NERNUNI)
IF (NER) 45, 48, 45
45 NEX = 45
GO TO 50
48 CONTINUE
KDIF = 0
KEQ = 0
DO 55 K = -14000
IF (N1(K) - N2(K)) 50, 52, 50
50 KDIF = KDIF + 1
GO TO 55
52 KEQ = KEQ + 1
...
55 CONTINUE
RITE
(43) KOI KEKDI - F EQ
43 FORMAT (5X, 14H PASS 1 KDI = , I5, 5X, 6H KE = , I5)
...
60 CONTINUE
REWIND AND EXCHANGE UNITS FOR COMPLETE NEW CYCLE
... END

VARIABLE ASSIGNMENTS FOR ROUTINE BUF
NI = BUFWT(0) N2 = 00000C00 NN = 017500C00 NU1 = 0002017 VU2 = 000216
NMAX = BUFWT(500) I = 000213 NER = 000212 NUNI = 000211 KP = 000210
J = BUFWT(16) KEQ = 000205 KDI = 000204 K = 000203 KDIF = 000202
END

COMPILE TIME = 473 MILLISECS
TOTAL CORE USED 064214

ENTRY POINT LOCATION ROUTINE ORIGIN
BUF 061700 061700
EXIT 062155 062155
END 062155 062155
STOP 062155 062155
BUFCL 062155 062155
BUFWT 062155 062155
HUFF 062155 062155
HUFFA 062155 062155
EXIT 063172 063172
BUFCL 063647 063647

COMMON BLOCKS LOCATION 000300
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>360</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>380</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1900</th>
<th>1901</th>
<th>1902</th>
<th>1903</th>
<th>1904</th>
<th>1905</th>
<th>1906</th>
<th>1907</th>
<th>1908</th>
<th>1909</th>
<th>1910</th>
<th>1911</th>
<th>1912</th>
<th>1913</th>
<th>1914</th>
<th>1915</th>
<th>1916</th>
<th>1917</th>
<th>1918</th>
</tr>
</thead>
<tbody>
<tr>
<td>1920</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1940</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1960</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1980</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2060</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 3920 |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 3940 |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 3960 |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| 3980 |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| PASS 1 KOI = 0 KE = 4000 PASS 2 KOIF = 0 KEQ = 4000 PASS 3 KOI = 0 KE = 4000 PASS 4 KOIF = 0 KEQ = 4000 PASS 5 KOI = 0 KE = 4000 PASS 6 KOIF = 0 KEQ = 4000 PASS 7 KOI = 0 KE = 4000 PASS 8 KOIF = 0 KEQ = 4000 PASS 9 KOI = 0 KE = 4000 PASS 10 KOIF = 0 KEQ = 4000 PASS 11 KOI = 0 KE = 4000 PASS 12 KOIF = 0 KEQ = 4000 PASS 13 KOI = 0 KE = 4000 PASS 14 KOIF = 0 KEQ = 4000 PASS 15 KOI = 0 KE = 4000 PASS 16 KOIF = 0 KEQ = 4000 PASS 17 KOI = 0 KE = 4000 PASS 18 KOIF = 0 KEQ = 4000 PASS 19 KOI = 0 KE = 4000 PASS 20 KOIF = 0 KEQ = 4000
<table>
<thead>
<tr>
<th>P</th>
<th>000000</th>
<th>RA</th>
<th>020000</th>
<th>FL</th>
<th>064214</th>
<th>EM</th>
<th>070000</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>002707</td>
<td>X0</td>
<td>77777777777777000000</td>
<td>B0</td>
<td>000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>002736</td>
<td>X1</td>
<td>00000000000000000000</td>
<td>B1</td>
<td>000001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>002663</td>
<td>X2</td>
<td>00000000000000102100</td>
<td>B2</td>
<td>000017</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>002671</td>
<td>X3</td>
<td>00000000000000000000</td>
<td>B3</td>
<td>000055</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>000021</td>
<td>X4</td>
<td>00000000000000000000</td>
<td>B4</td>
<td>000017</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>000003</td>
<td>X5</td>
<td>00000000000000000000</td>
<td>B5</td>
<td>000012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>002663</td>
<td>X6</td>
<td>02000000020000000000</td>
<td>B6</td>
<td>063276</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td>000001</td>
<td>X7</td>
<td>00000000000000000000</td>
<td>B7</td>
<td>000465</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CPU TIME = 42 SECONDS**

---

**PAGE LIMIT EXCEEDED**

---

---
DEBUG

Purpose This is a debugging routine that may be used during program execution to print the panel and a selective dump in any of the following three ways:

1) On the P count (program instruction count).
2) On the occurrence of a variable name as an operand in the object code.
3) On the appearance of branch instructions in the object code between certain instructions.

Caution: Introducing DEBUG slows the program down about 100 to 1. A program that runs 70 seconds without DEBUG will run about 7000 seconds with DEBUG.

Fortran CALL DEBUG

The call reads the snap cards which are located in the appropriate place in the data set and forms an address table with this information. If the call to DEBUG precedes the first read, the snap cards should be the first data cards in the deck. Snap cards must appear in the data set wherever the call to DEBUG occurs. DEBUG need be called only once.

Entry Point DEBUG

Ascent Calling Sequence RJ DEBUG

Error Messages An error message will be printed if CM or DEBUG is violated by a LOAD or STORE not within range. The program terminates.
Library Routines - Utility

Error Messages (cont'd)

Addresses on snap cards outside the field length of your program are ignored. If the address is out of bounds, the dump stops at the field limit address. No message is printed.

Storage

7278 locations

Snap Cards

All snap cards are followed by an END card used to flag the last card in the snap set.

P Card

Format:

Col. 12345678...

\[
\begin{align*}
P & \text{ IDENT} & \text{SSSSSS} & \text{LLLLL} & \text{UUUUU} \\
P & \text{ IDENT} & \text{SSSS} & \text{LLLL} & \text{UUUUU} \\
P & \text{ IDENT} & \text{SSSS} & \text{LLLL} & \text{UUUUU} \\
\text{END} \\
\end{align*}
\]

Col. 1: P Indicates a snap on the P count during program execution.

Col. 2: blank

Col. 3-10: IDENT

Col. 11: SSSSSS LLLLLL UUUUUU

SS LLLL UUUU

SSSSSS

LLL LLL UUUU

The identifier printed to label which snap dump has occurred.

The snap dump addresses always start in Col. 11. They are 1- to 6-character addresses separated by one blank.

Snap dump when the P count reaches this address.

Dump from this lower address to this upper address.
Snap Cards
(cont'd)

U Card

Format:
Col. 12345678...

U IDENT      SSSSSS LLLLLL UUUUUU*
U IDENT      SSSS LLLL UUUU
U IDENT      SS LLL UUUU
END

Col. 1: U      Indicates an operand address (address of variable name) on which to dump.
Col. 2: blank
Col. 3-10: IDENT
Col. 11: SSSSSS LLLLLL UUUUUU
            SSSS LLLL UUUU
            SSSSSS
            LLLLLL
            UUUUUU

*If S=0 or 1, the snap does not work correctly.

T Card

Format:
Col. 12345678...

TZXXXXXX LLLLLL UUUUUU
TZXXXXXX LLLL UUUU
END

Col. 1: T      Requests a trace on branch operation codes.
Col. 2: Z      If Z ≠ blank, a trace is performed on all branches.
               If Z = blank, a trace is performed only on those operations selected by non-blank characters in the X-field.
Library Routines - Utility

Sample Program

See the following example. The trace card has a 1 in Col. 2, so all branches are searched between program instructions from 3000 to 3005.

The first trace line means that 01 branches were taken from 3002 to 3120 (the entry point for INPUTC), 3004 to 3120, and 3005 to 3120. There were no other branch instructions in this range.

The SNAP2 U card used the operand 3066 (3000 + 66 is R in the program) at location 3027. At 3027, there was a SA7 R instruction, thus there was a snap dump at this point on operand R. The operand also appeared in KODER (3727) where another snap dump on operand R occurred.

The SNAP1 dump occurred at program location 3051 as specified on the P card. The dump was from 3000 to 3110.

Before making up the snap cards, a *FORTRAN,L is needed to get the correct locations from the program listing.
PROGRAM TEST

C=COMPLEX C+u+R
INTEGER A+Z
CALL DEBUG

READ (5110) A

FORMAT (F10.1)

C=(1,0,2,0)

A=2.3
B=3.4

DECMPLX(A+B)

PRINT 100+G+D

FORMAT (*.COMPLEX*4F10.2)
E=AIMAG(D)
F=REAL(D)
R=CONJG(C)
PRINT 10+1+E+F+R

FORMAT (*.AIMAG+REAL+CONJG*4F10.1)

LOC FUNCTION

Y=4.635
X=LOC(Y)
Z=LOC(C)
PRINT 102+X+Z

FORMAT (*.LOC*2110,2021)

GO TO 8

END
<table>
<thead>
<tr>
<th>A/IMAG = REAL, CONJ</th>
<th>J, K</th>
<th>1, 0-2, 0</th>
</tr>
</thead>
</table>
| SNAP1 003051 | A0 05541 | X0 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Purpose

This routine, written in Ascent, calls a peripheral processor program which generates an octal core dump and then calls EXIT. A *TRAP card must be included among the monitor control cards if DUMP is called. DUMP output is always sent to the dd80.

Fortran

CALL DUMP

Entry Point

DUMP

Ascent Calling Sequence

RJ DUMP

A RJ to DUMP is taken. There are no options and no argument list.

Error Message

PERIPHERAL DUMP FROM LOC

Storage

138 locations
Purpose

This routine writes an end-of-file on the tape unit specified.

Fortran

END FILE i

Entry Point

ENDFIL

Ascent Calling Sequence

SAl  I
RJ   ENDFIL

X1 contains the unit number.

Storage

7 locations
# Library Routines - Utility EXIT

**EXIT**

<table>
<thead>
<tr>
<th>Purpose</th>
<th>To terminate a program or subprogram with a &quot;normal&quot; exit.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>CALL EXIT</td>
</tr>
<tr>
<td></td>
<td>STØP</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
<tr>
<td>Entry Points</td>
<td>EXIT, END, STØP</td>
</tr>
<tr>
<td>Error Message</td>
<td>None</td>
</tr>
<tr>
<td>Storage</td>
<td>2 locations</td>
</tr>
<tr>
<td>Method</td>
<td>A jump to cell 7 is taken if there is a CALL EXIT, END, or</td>
</tr>
<tr>
<td></td>
<td>STØP card in the program.</td>
</tr>
</tbody>
</table>
GBYTE

Purpose
These routines give formatting capability in terms of numbers of bits rather than numbers of characters.

Entry Points
GBYTE, GBYTES, SBYTE, SBYTES

Fortran
CALL GBYTE(NPACK, ISAM, IBIT, NBITS)

Ascent Calling Sequence
RJ
EQ *
CON NPACK
CON ISAM
CON IBIT
CON NBITS

Storage
1168 locations

Timing
About 18 μs per byte (about 3.5 times as fast as ENCODE-DECODE for handling 6-bit characters). It is about 7 times as fast as ENCODE-DECODE when the latter uses an I4 format.

Entry GBYTE
Call GBYTE(NPACK, ISAM, IBIT-offset, NBITS in byte) to get NBITS started at bit number IBIT (counting 0-59 from left) in word NPACKED.

GBYTE counts → 0 1 2 . . . . 59
bit position 59 58 57 . . . . . . . . . 2 1 0

Put the byte into ISAM right-adjusted, with the rest of cell ISAM set to zero. The maximum byte size is 60. The permissible range of IBIT-offset is 0-59.
Library Routines - Utility

GBYTE (cont'd)

Example

CALL GBYTE(NPA,ISA,0,6)

```
NPA 7700000000000000000000000000000
ISA 0000000000000000000000000000077
```

Entry SBYTE

Call SBYTE(NPACK,ISAM,IBIT-offset,NBITS in byte) to reverse the process that occurred in GBYTE. The byte size in NPACK is cleared and the byte put in; data surrounding the packed byte is not affected.

Example

```
DIMENSION NA(2)

CALL SBYTE(NA,NB,56,6)
```

```
NB  0000000000000000000000000000077

* NA(1),NA(2)  0000000000000000000000000000017|6000000000000000000000000
```

* In binary these two words would look like this:

```
NA(1)     NA(2)
binary    0 . . . 0000001111  1100000000 . . . 0
octal     0 7 6 1 7 6
```
Since the offset is 56 bits and the byte size is 6 bits, the total number of bits (62) is greater than 1 word (60 bits). SBYTE then continues packing into the next word; i.e., word boundaries are ignored in packing bytes.

Call GBYTES(NPACK,ISAM,IBIT-offset,NBITS,NSKIP,ITER) to get an ITER number of bytes from NPACK into the ISAM array. The first byte is specified as in GBYTE. Then there is a skip of NSKIP bits. Then the next byte (of same size and same skip) starts, etc. The maximum byte size is 60 bits, but the skip can be over 60 bits long. The permissible range of IBIT-offset is 0-59.

Example

```
DIMENSION ISB(2)
CALL GBYTES(NPB,ISB,3,6,9,2)
```

Skip 3 bits as specified in the IBIT-offset, then take a 6-bit byte (the two 7's), skip 9 bits (the next three 0's) as specified in NSKIP, and do this twice (ITER=2).
Library Routines - Utility

GBYTE (cont'd)

Entry SBYTES

Call SBYTES (same arguments as GBYTES) to reverse the process that occurred in GBYTES.

Example

```
DIMENSION ISB(2)
CALL SBYTES(NPC,ISB,45,6,3,2)
```

<table>
<thead>
<tr>
<th>ISB(1),ISB(2)</th>
<th>59 0 59 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPC</td>
<td>000000000000000000000000000000077</td>
</tr>
</tbody>
</table>

45 bits (IBIT-offset) are skipped in NPC. Then the right-most 6 bits (NBITS) in ISB(1) are packed into NPC, 3 bits are skipped in NPC (NSKIP), and the process is repeated (ITER=2) from ISB(2).
Library Routines - Utility

INPUTB

Entry Points INPUTB, OUTPTB

Error Message TAPE ERROR IN INPUTB OR OUTPTB

Storage 12078 locations. (The size of IOB is large because of I/O buffers required.)

Entry INPUTB

This routine transfers information into storage in binary mode (parity odd) from logical unit i.

Fortran

READ TAPE 7,A,B,C (assume A is dimensioned for 100)

Ascent Calling Sequence

| SB2 | 7 |
| SB1 | 0 |
| RJ | INPUTB |
| SB2 | 144B |
| SB1 | A |
| RJ | INPUTB |
| SB2 | BO |
| SB1 | B |
| RJ | INPUTB |
| SB2 | BO |
| SB1 | C |
| RJ | INPUTB |
| SB1 | -1 |
| RJ | INPUTB |

The unit number is put in B2, B1 is set to zero to signal the first pass into the routine, and a RJ to INPUTB is taken. A separate jump to INPUTB is taken for each variable on the list, where B1 is set to the address of the variable and B2 is set to the dimension if the variable is an array, or B2 = 0 if it is single-valued. When the last parameter of the list is reached, B1 is set to -1 and signals this RJ to INPUTB as the last pass into the subroutine.
Library Routines - Utility

INPUTB (cont'd)

Entry OUTPTB

Purpose
This routine transfers information from storage locations specified in the list to a specified logical unit in binary mode (odd parity).

Fortran
WRITE TAPE 9,(Z(I),I=I1,I2)

Ascent Calling Sequence

| SB2 | 9   | Tape unit |
| SB1 | 0   | B1=0 signals first entry |
| RJ  | OUTPTB |
| SA1 | I1  | Initialize I |
| BX7 | X1  |
| SA7 | I   |
| LOC | SA2 | I   |
| SB1 | X2  | Loop on I for each element of Z |
| SB1 | B1+Z-1 |
| SB2 | 0   |
| RJ  | OUTPTB |
| SA3 | I   |
| SX0 | 1   |
| SA4 | I2  |
| IX6 | X3+X0 |
| SA6 | A3  |
| IX5 | X4-X6 |
| PL  | X5,LOC |
| SB1 | -1  | Last entry to OUTPTB |
| RJ  | OUTPTB |

In this case the indexing is explicit and a separate call to OUTPTB is generated for each element in a loop over I. This I/O statement call is slower than the implicit call in the example under INPUTB above where only one call is generated for the array A dimensioned for 100.
INPUTC

Purpose

INPUTC is a central processor Fortran data input routine which is entered each time a READ n,L, READ INPUT TAPE, or READ(n,FMT)L statement is executed. This routine processes the input list and the FORMAT statement associated with the READ statement for that list.

The data passes from the external device through an I/O buffer into central memory, where it is converted into display code one record at a time. Each record is then assembled in central memory according to the format specification and transmitted to the location specified by the compiler for the list.

Storage

3418 locations

Fortran

READ(n,FMT)LIST or READ INPUT TAPE n,Format,LIST

Entry Point

INPUTC (uses subroutine KRAKER as an external)

Ascent Calling Sequence

For READ (9,100)A,B:

SB3  Format address 100
SB2   9
SB1   BO
RJ    INPUTC
SB2   BO
SB1   A
RJ    INPUTC
SB2   BO
SB1   B
RJ    INPUTC
SB1   777776B
RJ    INPUTC
INPUTC (cont'd)

Ascent Calling Sequence (cont'd)

B3 contains the reference to the format statement; B2 contains the tape number. B1 is zero and flags the first entry into the routine, and a RJ to INPUTC is taken. The list is then processed, the address of the variable is loaded into B1 (B2 is the dimension of the variable and equals zero if it is not dimensioned), and a RJ is taken. A separate RJ is generated for each item on the list. After the last item in the list is processed B1 is set to -1 to flag the last pass into the subroutine.

An implicit loop for input, such as READ(5,100)A, where A is dimensioned for N, is faster than READ(5,100)(A(I),I=1,N). One call is needed to INPUTC for A in the first case; where in the second, N calls to INPUTC are generated by the compiler, one for each A(I).

Error Messages

END OF FILE ON DATA INPUT
ILLEGAL USE OF LETTER
PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
EXCEEDED RECORD SIZE
ILLEGAL DATA ENCOUNTERED ON A READ
ZERO FIELD WIDTH ON A FORMAT SPEC
MULTIPLE DECIMAL POINTS IN DATA
EXponent TOO LARGE ON DATA INPUT
TAPE ERROR DURING READ
INTEGER TOO LARGE
DATA OVERFLOW EXP OUT OF RANGE
HOLLERITH FORMAT WITH LIST
INPUTS

Purpose

INPUTS is a central processor routine which is entered each time a DECODE statement is executed. The routine processes the DECODE list and the FORMAT statement associated with the DECODE statement. The data, which is in display code, is assembled in central memory according to the FORMAT specification, one record at a time. The data is then transmitted to the location specified by the compiler for the list.

Fortran

DECODE(c,n,V),L

Entry Point

INPUTS (uses subroutine KRAKER as an external)

Ascent Calling Sequence

For DECODE (10,103,IB)A

| SB4 | 10  | Number of characters |
| SB3 | B0 + Format address (n) |
| SB1 | B0  | Flag first entry     |
| RJ  | INPUTS |
| SB1 | IB   |
| SB2 | B0   |
| RJ  | INPUTS |
| SB2 | B0   |
| SB1 | A    |
| RJ  | INPUTS |
| SB1 | -1   |
| RJ  | INPUTS |

Flag last entry

Error Messages

ILLEGAL USE OF LETTER IN FORMAT
PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
EXCEEDED RECORD SIZE
ILLEGAL DATA ENCOUNTER FROM LOC
ZERO FIELD WIDTH IN FORMAT SPEC
MULTIPLE DECIMAL POINTS IN DATA
INTEGER TOO LARGE
EXPONENT TOO LARGE ON DATA INPUT
**Error Messages (cont'd)**

- DATA OVERFLOW EXP OUT OF RANGE
- NO DECIMAL POINT FOLLOWING WIDTH
- HOLLERITH FORMAT WITH LIST

**Storage**

- \(274_8\) for INPUTS + \(607_8\) for KRAKER
IOPROC

Purpose
This routine performs the functions of buffer statements plus certain tape input/output functions which the buffer statements cannot handle.

Entry Points
RDTAPE, WRTAPE, IOWAIT, BSTAPE

Storage
2538 locations

Error Message
LOGICAL TAPE NO MUST BE 1 THRU 10

Note: Tape densities are specified on ASSIGN cards.

Entry RDTAPE
To read a record:

CALL RDTAPE (NUNIT, MODE, NTYPE, NADDR, NWDCNT)

NUNIT
A logical tape unit 1 through 10.

MODE
0 = even parity (BCD mode), no character conversion.
1 = odd parity (binary mode), no character conversion.
2 = even parity (BCD mode), conversion of external BCD to display code (or vice versa if writing a record).
3 = odd parity (binary mode), conversion as for 2.

NTYPE
0 or 1 = read (or write) a record in the NCAR system format.
2 = read a record from another computer. For physical records longer than 512 sixty-bit words, you will only get the first 512 words.
3 = read a record from another computer of any length. (This option does not allow character conversion.)

Note: Add 4 to any of the above options to ignore read parity error. The system will not try to correct this error for you. Thus a type 7 is the same as type 3 but with no reread on parity errors.
Library Routines - Utility

IOPROC (cont'd)

Entry RDTAPE (cont'd)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NADDR</td>
<td>Address of the first word in the record.</td>
</tr>
<tr>
<td>NWDCNT</td>
<td>Number of words to be read from the record, or maximum possible record size (or words to write).</td>
</tr>
</tbody>
</table>

Ascent Calling Sequence

<table>
<thead>
<tr>
<th>RJ</th>
<th>EQ</th>
<th>CON</th>
<th>NUNIT</th>
<th>MODE</th>
<th>NTYPE</th>
<th>NADDR</th>
<th>NWDCNT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDTAPE</td>
<td>*+6</td>
<td>NUNIT</td>
<td>MODE</td>
<td>NTYPE</td>
<td>NADDR</td>
<td>NWDCNT</td>
<td></td>
</tr>
</tbody>
</table>
To write a record:

CALL WRTAPE (same arguments as RDTAPE)

NTYPE is always assumed to be 0 on a write. Thus the records longer than 512 words will be segmented. A 565-word (60 bits/word) record is written on the tape as follows:

- record gap -

<table>
<thead>
<tr>
<th>512 60-bit words</th>
<th>0401 _8</th>
</tr>
</thead>
<tbody>
<tr>
<td>word count</td>
<td>1000 _8</td>
</tr>
<tr>
<td>system information</td>
<td></td>
</tr>
</tbody>
</table>

- record gap -

<table>
<thead>
<tr>
<th>53 60-bit words</th>
<th>7402 _8</th>
</tr>
</thead>
<tbody>
<tr>
<td>word count</td>
<td>65 _8</td>
</tr>
<tr>
<td>system information</td>
<td></td>
</tr>
</tbody>
</table>

Note: Odd parity (binary mode) should normally be used on tapes even if the data format is in BCD code. In even parity, for a word count under 64 or a multiple of 64, a tape frame is written with all zeroes (and the parity bit zero). When reread on our system, the word count will be wrong—on other machines a parity error is often generated.

To wait for the completion of a read or write:

CALL I\_WAIT(NUNIT,NSTATE,NWDS)

NUNIT As before
NSTATE Status:
0 = good read or write
1 = EOF (or end of tape on a write)
2 = redundancy on read or unable to write.
IOPROC (cont'd)

Entry IOWAIT

NWDS Number of words read or written.

You may call IOWAIT and get a good return before a unit has been used.

Ascent Calling Sequence

RJ IOWAIT
EQ *+4
C0N NUNIT
C0N NSTATE
C0N NWDS

Entry BSTAPE

To backspace a record:

CALL BSTAPE (NUNIT,NTYPE)

NUNIT As before.

NTYPE 0 or 1 = Use only if tape was made on this machine. Backspaces one logical record on a tape written by BUFFEI or CALL WRTAPE. This will backspace only one physical record if the tape was written with a Fortran WRITE statement.

2 or 3 = Backspace one physical record.

No call to IOWAIT is required after a backspace.

Ascent Calling Sequence

RJ BSTAPE
EQ *+3
C0N NUNIT
C0N NTYPE

Limitations

. Use only units 1 through 10 or the job will be terminated.

. After a read or a write, you must call IOWAIT before you read or write again from the same unit.

. In a call to RDTAPE or WRTAPE, the array size must be equal to or larger than the NWDCNT specified.
**KØDER**

<table>
<thead>
<tr>
<th>Purpose</th>
<th>KØDER is called by ØUTPTC/ØUTPTS to scan the format and convert the data to the format specified for output.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>No access.</td>
</tr>
<tr>
<td>Ascent Calling Sequence</td>
<td>No access.</td>
</tr>
<tr>
<td>Error Message</td>
<td>Refer to ØUTPTC and ØUTPTS.</td>
</tr>
<tr>
<td>Storage</td>
<td>1011\text{_8} locations</td>
</tr>
</tbody>
</table>
KRAKER is called by INPUTC/INPUTS to scan the format and convert the data to the format specified for input.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>KRAKER is called by INPUTC/INPUTS to scan the format and convert the data to the format specified for input.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>No access.</td>
</tr>
<tr>
<td>Ascent Calling Sequence</td>
<td>No access.</td>
</tr>
<tr>
<td>Error Messages</td>
<td>Refer to INPUTC and INPUTS.</td>
</tr>
<tr>
<td>Storage</td>
<td>6078 locations</td>
</tr>
</tbody>
</table>
OUTPTC

Purpose

OUTPTC is a central processor Fortran data output routine which is entered each time a PRINT, PUNCH or WRITE OUTPUT TAPE statement is executed. This routine processes the output list and associated FORMAT statement. The data in the list is converted from its internal to external representation. One display code record of information at a time is assembled in central memory according to the FORMAT specification. As each record is assembled, it is passed to a central I/O buffer where it is transmitted to an external device by other system routines.

Fortran

PRINT, PUNCH, WRITE OUTPUT TAPE 6, WRITE(6,100)

Entry Point

OUTPTC (uses KODER as an external)

Ascent Calling Sequence

For WRITE (6,100) A,B:

| SB3  | Format address |
| SB2  | 6              |
| SB1  | B0             |
| RJ   | OUTPTC         |
| SB2  | B0             |
| SB1  | A              |
| RJ   | OUTPTC         |
| SB2  | B0             |
| SB1  | B              |
| RJ   | OUTPTC         |
| SB1  | -1             |
| RJ   | OUTPTC         |

B3 contains the format address, B2 contains the tape number or its address, B1 = 0 flags the first entry into the routine, and a RJ to OUTPTC is generated. For each variable in the list, a separate return jump is generated where B2 = the dimension of the variable and B1 = the address. B2 = 0 if there are no dimensions. The final pass to OUTPTC is flagged by setting B1 = -1.
Ascent Calling Sequence (cont'd)

Note: Only one call to OUTPTC is generated if a dimensioned variable is output implicitly, such as by specifying only the base address of the array in B1 and the dimension of A in B2. However, if an explicit loop is programmed in Fortran, such as

```
WRITE (6,100)(A(I),I=1,N)
```

A RJ to OUTPTC is generated for each A(I). Thus N return jumps to OUTPTC would be taken in a Fortran compiled loop.

For PUNCH 102, (A(I),I=M,MM,10):

```
SB2    74B
SB3    Format address
SB1    0
RJ     OUTPTC
SA4    M
BX7    X4
SA7    I
LOC    SA1   I
SB1    X1
SB1    B1+A-1
SB2    0
RJ     OUTPTC
SA5    I
BX0    10
SA2    MM
IX6    X5+X0
SA6    A5
IX3    X2-X6
PL     X3,LOC
SB1    -1
RJ     OUTPTC
```

Notice that the punch unit is 74, and since the list indexing is explicit, a RJ to OUTPTC is done for each value of I.
Library Routines - Utility

OUTPTC (cont'd)

Error Messages

- Paren group not closed in format
- Illegal functional letter in format statement
- Zero field width in format statement
- Output record length exceeded
- Field width less than decimal width
- Tape error on output
- Hollerith format with list

Storage

4548 for OUTPTC + 10118 for KODER
OUTPTS

Purpose
OUTPTS is a central processor routine which is entered each time an ENCODE statement is executed. This routine processes the ENCODE list and associated FORMAT statement. The data in the list is converted from machine language representation to display code according to the format specification. One display code record of information is assembled in central memory and transmitted to a location specified by the ENCODE statement.

Fortran
ENCODE(c,n,V)L

Entry Point
OUTPTS (uses subroutine KODER as an external)

Ascent Calling
Sequence

| SB4 | BO + c |
| SB3 | BO + n (format address) |
| SB1 | BO |
| RJ  | OUTPTS |
| SB1 | V |
| SB2 | BO |
| RJ  | OUTPTS |
| SB1 | BO + L |
| SB2 | BO |
| RJ  | OUTPTS |
| SB1 | 777776B |
| RJ  | OUTPTS |

Error Messages
PAREN NOT CLOSED IN FORMAT
ILLEGAL FUNCTIONAL LETTER IN FORMAT
ZERO FIELD WIDTH IN FORMAT
OUTPUT RECORD LENGTH EXCEEDED
HOLLERITH FORMAT WITH LIST
FIELD WIDTH LESS THAN DECIMAL WIDTH

Storage
2768 for OUTPTS + 10118 for KODER
OVERLAY

Description

This subroutine provides a technique for bringing routines into the central memory from the peripheral drum, so that several routines will occupy the same storage locations at different times. OVERLAY is used when the total storage requirements for instructions exceed the available main storage. A program which exceeds the available memory is divided into independent parts which may be called and executed as needed. Such programs consist of a control program, overlays of the control program, and segments of the overlays.

Purpose

The overlay subroutine can be used to run as one job the series of jobs where program A generates output which is to be used as input to program B. The case may be that A and B are too large to fit in central memory together, or A must be completely processed before the data is ready for B. This procedure may run through any string of programs.

By constructing a control program to move programs A and B as overlays in a sequential manner to operate on a data set, it is possible to accomplish with the running of one deck what formerly took several decks and a great deal of time.

Possibilities can be extrapolated from this point to include input options for different integration schemes to be called as overlays, or for different output programs to be called as overlays.

Subroutine OVERLAY provides the user with an almost unlimited program size when the program is divided into overlays and segments. OVERLAY does not provide the user with unlimited data storage capabilities.
Definition of Terms

Control Program
A sequence of instructions which prescribes the series of steps to be taken in moving a set of "overlay programs" into core to perform their individual operations on a data set. A control program may or may not be considered as a main program since a given set of main programs could be used as overlays and moved by a control program.

Overlay Program
A sequence of instructions which performs an operation on data as would a subroutine. An overlay program is moved into core only upon command of the control program.

Segment Program
A sequence of instructions which perform an operation on data as would a subroutine. The segment program is associated with only one overlay program and can only be called into core by this overlay program.

Overlay Identification Card
The overlay identification card is used to flag the loader as to what kind of program is currently being processed and with which program it is to be associated. This card is placed immediately after the PROGRAM NAME or SUBROUTINE NAME card and has one of the following formats:

1) \( \text{OVERLAY}(I\emptyset,0) \) - the program identification card for an overlay program, where
\[ I\emptyset = \text{overlay number} \]
\[ 0 = \text{segment number}, 0 \]
Definition of Terms (cont'd)

Overlay Identification Card (cont'd)

2) \texttt{OVERLAY(I\emptyset,IS)} - the overlay identification card for a segment program, where
   \begin{align*}
   I\emptyset &= \text{overlay program number with which this segment program as associated} \\
   IS &= \text{segment number}
   \end{align*}

3) \texttt{OVERLAY(I\emptyset,IS,NAME)} - This overlay identification card is used with an overlay or a segment program and functions in two ways.
   a) It is used when a Binary deck is to be punched and subsequently loaded with non-binary decks.
   b) It is used when the overlay or segment program contains dimensioned variables not in Blank or labeled Common.

NAME is the entry point name of the overlay or segment program. The entry point name is the NAME on the SUBROUTINE NAME card associated with the given overlay or segment program. I\emptyset and IS are the same as above.

Overlay Call Statements

The Fortran call statement for an overlay or segment program has one of the following formats:

1) \texttt{CALL OVERLAY(I\emptyset,IS)} where
   \begin{align*}
   I\emptyset &= \text{overlay number} \\
   IS &= \text{segment number when applicable}
   \end{align*}

2) \texttt{CALL OVERLAY(I\emptyset,IS,LIST)} where
   \begin{align*}
   I\emptyset &= \text{overlay number} \\
   IS &= \text{segment number when applicable} \\
   LIST &= \text{parameter LIST(I,A) that appeared on the SUBROUTINE NAME(I,A) card.}
   \end{align*}
Rules

Control Program

- The control program may contain subprograms.
- The control program always remains in central memory during execution.
- The control program may only call overlays.
- The control program is identified by an OVERLAY(0,0) card.
- Subroutines which are not from the system but loaded with the control program are identified with an OVERLAY(0,0) card.
- All subroutines loaded with the control program may be called from the control program, overlay program, segment program, or other subroutine.

Overlay Programs

- An overlay program is a subprogram to the control program.
- Overlay programs are loaded from the drum when called by the control program.
- Overlay programs may only call their associated segment programs.
- An overlay program may not call another overlay program.
- An overlay program is identified by an OVERLAY(I0,0) card.
- An overlay program is called into the central processor with a CALL OVERLAY(I0,0,LIST) type of call statement.
- An overlay program may have subroutines associated only with it. These subroutines can only be called by this overlay program and its associated segments.
- The subroutines associated only with a given overlay program are identified with an OVERLAY(I0,0,NAME) type of card. NAME is the overlay entry point name. These subroutines are called in the ordinary way in the overlay.
- An overlay program may call any subroutine associated with the control program.
OVERLAY (cont'd)

Rules (cont'd)

Segment Programs

. A segment program is a subprogram to an overlay program.

. A segment program cannot be used as a subprogram to the control program.

. Segment programs are loaded from the drum when called by their associated overlay program.

. Segment programs may not call the control program, any overlay program, or any other segment program.

. A segment program is identified by an OVERLAY(IO,IS) card.

. A segment program is called into the central processor with a CALL OVERLAY(IO,IS,LIST) type of call statement.

. A segment program may have subroutines associated uniquely with it. These subroutines can only be called by this segment program.

. The subroutines uniquely associated with a given segment are identified with an OVERLAY(IO,IS,NAMES) type of card. NAME is the segment entry point NAME. These subroutines are called in the ordinary way in the segment.

. A segment program may call any subroutine associated with its overlay program or the control program.

General

. Only the control program, one overlay, and one segment may occupy central memory at a given time.

. A LIST may be transferred to an overlay or segment program. The preferred means of variable transfer is through blank common.
Central Memory Allocation

<table>
<thead>
<tr>
<th>Location</th>
<th>Central memory allocation is as follows:</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>Central monitor</td>
</tr>
<tr>
<td>03000</td>
<td>Overlay control table</td>
</tr>
<tr>
<td>03120</td>
<td>Blank common</td>
</tr>
<tr>
<td>Loc 1</td>
<td>Control program and dimension</td>
</tr>
<tr>
<td></td>
<td>+ major subroutine</td>
</tr>
<tr>
<td></td>
<td>+ system subroutines (e.g., SIN, COS, PLOT, etc.)</td>
</tr>
<tr>
<td>Loc 2</td>
<td>Labeled common</td>
</tr>
<tr>
<td>Loc 3</td>
<td>Overlay origin</td>
</tr>
<tr>
<td>Loc 4</td>
<td>Segment origin</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

End Allocation

Loc 1 through Loc 4 are arbitrary core locations depending on the relative size of the areas.
There are some cases to consider if different kinds of programs are loaded together with subroutine OVERLAY.

**Fortran**

Fortran decks immediately after the control program will be loaded with it unless overlay cards signal otherwise. Decks may be placed out of order but loaded with any overlay if the proper overlay card is present.

**Ascent**

An Ascent program may be used as an overlay or a segment. An overlay card is placed immediately after the Ascent card and has the following format:

```
Col. 11: OVERLAY
Col. 20: IØ,IS
```

where IØ and IS are defined above.

**Cosy**

The Cosy deck must have been punched from an Ascent deck which contained the proper overlay card.

**Binary**

The binary deck must have been punched from a Fortran or Ascent deck which contained the proper overlay card. If a binary deck is being loaded with non-binary decks, then additional rules take effect since binary decks are loaded first:
Program Loading (cont'd)  
Binary (cont'd)  

1) If the binary deck is the control program or used by the control program, then an \texttt{OVERLAY(0,0)} card must have been used when the deck was punched.

2) If the binary deck is an overlay or a segment, then an \texttt{OVERLAY(IO,IS)} card must have been used when the deck was punched.

3) If the binary deck is a subroutine to be loaded with an overlay or a segment, then an \texttt{OVERLAY(IO,IS,NAME)} card must have been used, etc. \texttt{NAME} is the name of the entry point to the overlay with which this particular binary subroutine is associated.

Sample Program  
The following program is included together with an explanation of the loader map to illustrate the use of \texttt{OVERLAY}.
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>*FORTRAN</td>
</tr>
<tr>
<td>00000</td>
<td>PROGRAM CONTROL</td>
</tr>
<tr>
<td>00000</td>
<td>OVERLAY (0:0)</td>
</tr>
<tr>
<td>00000</td>
<td>COMMON A(10), B(10), C(10), D(10)</td>
</tr>
<tr>
<td>00000</td>
<td>DIMENSION E(10)</td>
</tr>
<tr>
<td>00001</td>
<td>XI=0.0</td>
</tr>
<tr>
<td>00002</td>
<td>XA=1.0</td>
</tr>
<tr>
<td>00002</td>
<td>DO 50 I=1,10</td>
</tr>
<tr>
<td>00002</td>
<td>SUM=D+0</td>
</tr>
<tr>
<td>00002</td>
<td>C</td>
</tr>
<tr>
<td>00002</td>
<td>DO 10 J=1,10</td>
</tr>
<tr>
<td>00002</td>
<td>A(J)*XI</td>
</tr>
<tr>
<td>00002</td>
<td>XI=XI*XA</td>
</tr>
<tr>
<td>00003</td>
<td>10 CONTINUE</td>
</tr>
<tr>
<td>00003</td>
<td>CALL 1ST OVERLAY = CALCULATION PHASE</td>
</tr>
<tr>
<td>00004</td>
<td>CALL OVERLAY (1=0+XI+XA)</td>
</tr>
<tr>
<td>00005</td>
<td>CALL OVERLAY (2=0)</td>
</tr>
<tr>
<td>00006</td>
<td>CALL OVERLAY (3=0+X)</td>
</tr>
<tr>
<td>00007</td>
<td>CALL OUTPUT PROGRAM</td>
</tr>
<tr>
<td>00007</td>
<td>CALL EXIT</td>
</tr>
<tr>
<td>00007</td>
<td>CALL COS</td>
</tr>
<tr>
<td>00007</td>
<td>CALL SIN</td>
</tr>
<tr>
<td>00007</td>
<td>END</td>
</tr>
</tbody>
</table>

VARIABLE ASSIGNMENTS FOR ROUTINE CONTROL

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>00000</td>
</tr>
<tr>
<td>A</td>
<td>00000</td>
</tr>
<tr>
<td>B</td>
<td>00000</td>
</tr>
<tr>
<td>C</td>
<td>00000</td>
</tr>
<tr>
<td>D</td>
<td>00000</td>
</tr>
<tr>
<td>XI</td>
<td>00000</td>
</tr>
<tr>
<td>XA</td>
<td>00000</td>
</tr>
<tr>
<td>I</td>
<td>00000</td>
</tr>
<tr>
<td>SUM</td>
<td>00000</td>
</tr>
<tr>
<td>J</td>
<td>00000</td>
</tr>
</tbody>
</table>

COMPILE TIME = 175 MILLISECS
```
SUBROUTINE OVLY10 (YI, YA)
OVERLAY (IT0)
C THIS IS THE FIRST OVERLAY
COMMON A(lO), B(lO), C(lO), D(lO)
YC = YI * YA
.. ...
CALL OVERLAY (1x1, YC)
CALL OVERLAY (1x2, YS)
RETURN
END

CALL 1ST SEGMENT TO THIS OVERLAY - FORM B
CALL 2ND SEGMENT TO THIS OVERLAY - FORM C

VARIABLE ASSIGNMENTS FOR ROUTINE OVLY10
YI - 000000, YA - 000000, A - 000000, B - 000000, C - 000024, D - 000036

COMPILE TIME = 70 MILLISECS

SUBROUTINE SEG11 (Z)
OVERLAY (l1)
C THIS IS THE 1ST SEGMENT OF THE 1ST OVERLAY
COMMON A(lO), B(lO), C(lO), D(lO)
C COMPUTE AN ARRAY B
DO 10 I = 1, 10
B(I) = SINF (A(I) - (.5*Z))
10 CONTINUE
RETURN
END

VARIABLE ASSIGNMENTS FOR ROUTINE SEG11
Z - 000000, A - 000000, B - 000000, C - 000024, D - 000036, I - 000022

COMPILE TIME = 60 MILLISECS

SUBROUTINE SEG12 (S)
OVERLAY (l2)
C THIS IS THE 2ND SEGMENT OF THE 1ST OVERLAY
COMMON A(lO), B(lO), C(lO), D(lO)
C COMPUTE AN ARRAY C
DO 20 I = 1, 10
C(I) = COSF (A(I) - (.5*S))
20 CONTINUE
RETURN
END

VARIABLE ASSIGNMENTS FOR ROUTINE SEG12
S - 000000, A - 000000, B - 000000, C - 000024, D - 000036, I - 000022

COMPILE TIME = 63 MILLISECS
000000 SUBROUTINE OVLY20
000000 OVERLAY (2TO)
000000 C THIS IS THE SECOND OVERLAY
000000 COMMON A(10),B(10),C(10),D(10)
000000 C COMPUTE AN ARRAY D
000000 DO 10 I=1,10
000002 D(I)= 2.0*(A(I)-B(I)+C(I))
000010 CONTINUE
000012 RETURN
000013 END

VARIABLE ASSIGNMENTS FOR ROUTINE OVLY20
A = 00000000 B = 00001000 C = 00002000 D = 00003000 I = 00004000

SUBROUTINE OUTPUT (E)
000000 OVERLAY (3TO)
000000 C THIS IS THE THIRD OVERLAY
000000 DIMENSION E(10)
000000 WRITE (6*10)(E(I),I=1,10)
000010 FORMAT (1I1,1X,10(1X,E10.3))
000012 RETURN
000014 END

VARIABLE ASSIGNMENTS FOR ROUTINE OUTPUT
E = 00000000 I = 00002000

COMPILE TIME= 49 MILLISECS
<table>
<thead>
<tr>
<th>TOTAL CORE USED</th>
<th>005337</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL PROGRAM LENGTH</td>
<td>003734</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>LOCATION</th>
<th>ROUTINE ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL</td>
<td>003202</td>
<td>003170</td>
</tr>
<tr>
<td>EXIT</td>
<td>003273</td>
<td>003273</td>
</tr>
<tr>
<td>END</td>
<td>003273</td>
<td>003273</td>
</tr>
<tr>
<td>STOP</td>
<td>003273</td>
<td>003273</td>
</tr>
<tr>
<td>SIN</td>
<td>003302</td>
<td>003276</td>
</tr>
<tr>
<td>COS</td>
<td>003276</td>
<td>003276</td>
</tr>
<tr>
<td>OVERLAY</td>
<td>003371</td>
<td>003302</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMMON BLOCKS LOCATION</th>
<th>003120</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVERLAY</td>
<td>000001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>LOCATION</th>
<th>ROUTINE ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>OVLY10</td>
<td>003574</td>
<td>003574</td>
</tr>
<tr>
<td>GEDR5D</td>
<td>003631</td>
<td>003631</td>
</tr>
</tbody>
</table>

| SEGMENT | 000001 | 000000 | LOCATION 003635 | LENGTH 000027 |

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>LOCATION</th>
<th>ROUTINE ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEG11</td>
<td>003635</td>
<td>003635</td>
</tr>
<tr>
<td>SEG12</td>
<td>003635</td>
<td>003635</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENTRY POINT</th>
<th>LOCATION</th>
<th>ROUTINE ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>OUTPUT</td>
<td>003574</td>
<td>003574</td>
</tr>
<tr>
<td>GEDR5D</td>
<td>003626</td>
<td>003626</td>
</tr>
<tr>
<td>KEDR</td>
<td>003650</td>
<td>003650</td>
</tr>
<tr>
<td>OUTFCT</td>
<td>004662</td>
<td>004662</td>
</tr>
</tbody>
</table>

| | 2.672E+01 | 5.061E+01 | 8.366E+01 | 1.090E+02 | 1.334E+02 | 1.603E+02 | 1.926E+02 | 2.153E+02 | 2.401E+02 | 2.720E+02 |

CPU TIME = 1 SECONDS

**************************************************************************
**************************************************************************
The loader map follows the usual format to where the first overlay is loaded. The first overlay and its first segment are discussed below.

a) OVERLAY 000001 000000

This represents the overlay card placed before the SUBROUTINE OVLY10 card and read OVERLAY(1,0).

b) LOCATION 003574 LENGTH 000057

This part of the line of information gives the location at which OVERLAY(1,0) will be loaded when it is called by the control program. It will occupy the next 57 cells.

c) ENTRY POINT LOCATION ROUTINE ORIGIN
   OVLY10 003574 003574
   Q8QRSD 003631 003631

The first entry point is the name of the subroutine which was labeled as OVERLAY(1,0). The location of this entry point follows. The last number is the origin of the routine this entry point is in. The routine had two entry points, OVLY10 and Q8QRSD.

d) SEGMENT 000001 000001

This represents the overlay card placed before the SUBROUTINE SEG11 card and read OVERLAY(1,1).
e) LOCATION 003653 LENGTH 000027

This part of the line of information gives the location at which this segment will be loaded when it is called by its associated OVERLAY. It will occupy the next 27 cells after the overlay is loaded.

f) ENTRY POINT LOCATION ROUTINE ORIGIN
SEG11 003653 003653 003653

The first entry point is the name of the subroutine which was labeled OVERLAY(1,1). The rest of the description is the same as for item c) above.
PAUSE

Purpose
PAUSE stops program execution in order that the operator may perform certain tasks such as setting sense switches. Program execution may then be resumed.

Method
PAUSE stops program execution with the words PAUSEn printed on the console along with the program's control point number under the heading EXECUTING JOBS. If n is zero, only PAUSE is printed. Instructions for the operator are written on the job card in the space provided.

Example of console printout:

```
EXECUTING JOBS
01 PAUSE
```

To restart execution, the operator types the request GO after typing the control point number, in this case 01, at the console.

Example:

```
01 GO
```

Fortran
PAUSE or PAUSEn

Entry Point
PAUSE

Ascent Calling Sequence

| SA2 | N |
| BX7 | X2 |
| RJ  | PAUSE |

In X7 the first 15 bits contain the PAUSE identifier (n). Only the first 5 characters of N will be used. A RJ to PAUSE is then taken.

Storage
208 locations
Library Routines - Utility

PDUMP

Purpose
PDUMP writes an octal dump of the entire program or parts of the program, depending on the arguments specified. This is a dump and proceed.

Entry Point
PDUMP

Storage
322 locations

Method
This is a central processor dump and proceed written in Ascent. The peripheral dump is not called.

Registers Saved
The A-registers are never saved. All other registers are saved and restored.

Variable Arguments
PDUMP uses a variable argument list. The following is a description of the four options, depending on 0, 1, 2, 3 arguments. The test for the argument count is made using the return jump address.

Call PDUMP (no arguments)
Ascent calling sequence:
   RJ       PDUMP
PDUMP will dump from 3000₈ to the field length (specified in FL on the panel).

Call PDUMP (3500B) (1 argument)
Ascent Calling Sequence:
   RJ       PDUMP
   EQ       *+2
   CON      NUM  (where NUM contains 3500B)

The starting address of the dump is the single argument, 3500B. A dump will be taken from this argument to the field length.
If the argument = 0, the dump starts at 3000₈.
Library Routines - Utility

PDUMP (cont'd)

Call PDUMP (3500B, 4000B) (2 arguments)

Ascent calling sequence:

\[
\begin{align*}
\text{RJ} & \text{ PDUMP} \\
\text{EQ} & \text{ *+3} \\
\text{CON} & \text{ NUM1} \quad \text{(contains 350B)} \\
\text{CON} & \text{ NUM2} \quad \text{(contains 400B)}
\end{align*}
\]

The starting address of the dump is in NUM1; the stopping address is in NUM2. If the first argument = 0, the dump starts at 3000; and if the second argument = 0, the dump terminates at the field length.

Example:

\[
\text{CALL PDUMP (0,4000B)}
\]

This dumps from 3000 to 4000.

Example:

\[
\text{CALL PDUMP (4500B,0)}
\]

This dumps from 4500 to field length.

Call PDUMP (A(1),B(30),1) (3 arguments)

Ascent calling sequence:

\[
\begin{align*}
\text{RJ} & \text{ PDUMP} \\
\text{EQ} & \text{ *+44} \\
\text{CON} & \text{ A} \\
\text{CON} & \text{ B} \\
\text{CON} & \text{ N} \quad \text{(where N = 1)}
\end{align*}
\]

The flag (N=1) that 3 arguments are present signals that the argument contains the actual address itself, not a reference to that address. Thus the dump in this case proceeds from A(1) to B(30). Another choice here would be to dump a whole routine or certain routines. For example, a CALL PDUMP(EXIT),
Library Routines - Utility

PDUMP (cont'd)

Variable Arguments (cont'd)  

Limitations  

Only the first 20 words of the area between 0 and 3000 will be dumped.

Example

CALL PDUMP(5,2500B)

This will dump from 5 to 20 only.
Purpose  This set of programs is designed to provide the user with the capability of writing, reading, and rewriting records of differing length, at random, to and from the peripheral drum. (Do not assign the drum, the system will do it for this routine.)

Entry Points  RANINT, RANWT, RANRD

Entry RANINT

Fortran  RANINT(ID,IDLEN,NERROR)

Description  The RANINT subroutine is the initializing routine and must be called first. This routine is used to set up certain tables and flags which are subsequently used by RANWT and RANRD.

The core space for these tables and flags must be supplied by the user and transmitted in the calling sequence of RANINT. The core space to be supplied is defined as follows:

ID  An array of at least five core cells for tables. The ID array will be divided into three tables to provide information for each record, namely:

First 1/5 of the array ID = TABLE 1  Name table. Each different record must have a unique identifying name.

Second 2/5 of the array ID = TABLE 2  Pointer table. Each named record has additional information in this table for reads and rewrites.

Third 2/5 of the array ID = TABLE 3  Drum information table.
IDLEN The length of the ID array. In order to supply enough ID TABLE area the user must provide five times the number of uniquely named records he plans to write.

NERROR An error flag cell checked by the user in his program after each call to RANINT, RANWT and RANRD. If NERROR = 0, all operations of the last call were successful. If NERROR ≠ 0, an error in the last call occurred. By printing NERROR in an 12 format, the user can check the NERROR number against the error flag table, which follows:

NERROR = 1: Given length of the ID array is less than 5.

= 2: Initial entry of CALL RANINT was not made.

= 3: Attempt to write a new record without giving a length.

= 4: Attempt to write a new record failed.

= 5: Attempt to write more than given ID table allotment.

= 6: Attempt to rewrite failed.

= 7: Called for a read before the first write was made.

= 8: Attempt to read a record not previously written.

= 10: Attempt to read failed.
RANRD (cont'd)

Entry RANWT

Fortran

RANWT(NAME, ARRAY1, NERROR, LENGTH)
RANWT(NAME, ARRAY1, NERROR)

Description

The RANWT subroutine provides two calling sequences for writing records. The first calling sequence has one extra argument, LENGTH, and is used for writing a given record for the first time. The second calling sequence is used for rewrites of previously written records. Therefore, once a given record is written with the first calling sequence, that LENGTH is the one used for all subsequent reads and rewrites. The length of a given record will not be changed after the first time it has been written. The arguments are defined as follows:

NAME    A variable name containing one 60-bit word of any configuration, other than an indefinite, which is unique for each record. This name identifies this particular record.

ARRAY1  The array of data to be written.

ERROR   The error flag cell.

LENGTH  The number of words of ARRAY1 which is to be written. This is also the permanent length of the record with this NAME and will be used for all subsequent operations pertaining to this record.
Entry RANRD

Fortran

RANRD(NAME,ARRAY2,NERROR)

Description

The RANRD subroutine reads the data identified by NAME into ARRAY2 with the length given when the record was previously written. The arguments are defined as follows:

- **NAME**: The record identifier as described in entry RANWT.
- **ARRAY2**: The array into which the data is read.
- **NERROR**: The error flag cell.

Rules

To use this set of routines properly the following rules must be observed:

- Call RANINT first and only once.
- Call RANWT (4 arguments) for new record writes.
- Check the error flag after each call.
- Special provisions must be made at RESTART time when using the SAVE routine together with the random read/write routines. All records must be rewritten using the original length after RESTART. Normal flow may then be entered.
PROGRAM RANREAD
DIMENSION A(100),B(100)
C IU MUST BE DIMENSIONED FIVE TIMES THE NUMBER OF UNIQUELY NAMED RECORDS WE PLAN TO WRITE
C THE CALL TO RANINT SETS POINTERS AND MUST PRECEDE ANY CALLS TO RANWT OR RANRD
CALL RANINT(ID(20),NERROR)
IF(NERROR.NE.0)GO TO 10
A(1)=1.0
DO 1 I=2,100
1 A(I)=A(I-1)+1.0
C ANY 60 BIT CONFIGURATION DEFINES THE RECORD NAMES
FIRST=18
SECOND=1.36E-0
THIRD=MARY
FOURTH=1.0
C ON THE FIRST CALL TO ANY UNIQUE RECORD THE RECORD LENGTH MUST BE INCLUDED
CALL RANWT(FIRST,A(1),NERROR,10)
IF(NERROR.NE.0)GO TO 11
CALL RANWT(SECOND,A(51),NERROR,20)
IF(NERROR.NE.0)GO TO 11
C SOMETHING THAT WAS PREVIOUSLY WRITTEN WITH RANWT MAY NOW BE BROUGHT IN WITH RANRD
CALL RANRD(FIRST,B(1),NERROR)
IF(NERROR.NE.0)GO TO 12
WRITE(6,115)(R(I),I=1,10)
CALL RANWT(THIRD,A(11),NERROR,5)
IF(NERROR.NE.0)GO TO 11
CALL RANWT(FOURTH,A(16),NERROR,25)
IF(NERROR.NE.0)GO TO 11
C AFTER THE FIRST CALL TO ANY UNIQUELY NAMED RECORD THE LENGTH OF THE RECORD TO BE WRITTEN MAY BE OMITTED
CALL RANWT(FIRST,A(51),NERROR)
IF(NERROR.NE.0)GO TO 11
CALL RANWT(FIRST,B(91),NERROR)
IF(NERROR.NE.0)GO TO 12
WRITE(6,115)(R(I),I=1,100)
WRITE(6,115)(A(I),I=1,100)
115 FORMAT(10(2XF10.3))
CALL EXIT
50 CALL EXIT
12 WRITE(6,112)NERROR
112 FORMAT(/20X'ERROR ON RANRD*')
10 TO 50
11 WRITE(6,111)NERROR
111 FORMAT(/20X'ERROR ON RANWT*')
GO TO 50
SAVEF

Fortran

FARG=SAVEF(ISAV)
FARG=SAVEF(ISAV,IOVLY)
FARG=SAVEF(ISAV,IOVLY,IHISW)

Description

The SAVEF subroutine is used in jobs requiring more than 1/4 hour of machine time. The purpose of the routine is to save on logical unit 3 the complete internal state of central memory whenever SAVEF is called. SAVEF also makes provision for saving all overlays used with the program and allows the user to write a data file on the end of the save tape.

The advantages of such a routine are obvious for long-running programs. Standard procedure is to call SAVEF approximately every 15 minutes. This allows the job to be restarted in the event of a machine malfunction without losing more than 15 minutes of machine time.

There are three possible ways to call SAVEF:

1) FARG=SAVEF(ISAV)

This is the standard method of calling SAVEF where ISAV is either 0 or 1 and one sense switch setting is allowed as follows:

ISAV=0 If sense switch 6 is on, save the state of the computation on logical unit 3 and terminate execution. If sense switch 6 is off, continue without interruption.

ISAV=1 If sense switch 6 is on, save the state of the computation on logical unit 3 and terminate execution. If sense switch 6 is off, save the state of the computation on logical unit 3 and continue the computation.
SAVEF (cont'd)

Description (cont'd)
The setting of sense switch 6 allows the operator to stop a program containing a SAVEF call after a given length of real time, or in case of any emergency. This is provided that the programmer has made such allowances by checking for sense switch 6 at regular intervals. These intervals are less than the regular save intervals. The check is placed at a logical stopping point in case sense switch 6 is set.

2) FARG=SAVEF(ISAV,IOVLY)

This calling sequence is used when a program contains overlays and segments. (See description of subroutine OVERLAY.)

   ISAV
   ISAV
   ISAV
   Same as described in calling sequence 1) above.

   IOVLY=0
   IOVLY=0
   IOVLY=0
   Is a flag to the save program indicating that there are no overlays to be moved from the drum to the save tape.

   IOVLY=1
   IOVLY=1
   IOVLY=1
   Is a flag to the save program indicating there are overlays to be moved to the save tape.

3) FARG=SAVEF(ISAV,IOVLY,IHISW)

This calling sequence is used when a program needs to write an additional file of history data to be used for initialization purposes at restart time. Provision has been made in the SAVEF program to jump to a user-constructed program which will write a history file on logical unit 3. When the IHISW option is in effect, this calling sequence is used regardless of whether or not a program contains overlays. The IOVLY flag is set for the appropriate condition in the case being run.
SAVEF (cont'd)

Description
(cont'd)

- ISAV: Same as described in calling sequence 1) above.
- IOVLY: Same as described in calling sequence 2) above.
- IHISW=0: No additional information is to be added to the save tape.
- IHISW=1: Jump to a user program called WTHIS, which will write a logical file of data on the save tape starting at its current position.

Definitions and Rules

- The program which calls SAVEF must have a tape assignment to logical unit 3, such as *ASSIGN,BS023=3. BS023 is a tape in the user's name, assigned to him by the operator. This tape will be used to restart the job which has been saved on this tape.
- Logical unit 3 is reserved for the exclusive use of SAVEF.
- The operators must be supplied a restart deck.
- A restart deck consists of the same control cards as were used with the program which called SAVEF with the following changes:
  a) Replace the *RUN card with the *RESTART card.
  b) Add a *LIMIT,C=NNNNN card to specify the core reservation limit. This number is equal to or greater than the total core used by the run which made the save tape. The amount of core needed may be determined from the loader map when the job is first saved.
- A time limit and output limit are specified on a *LIMIT card in the restart deck.

Sample:

```
*JOB,5000,F0540004,SMITPH
*ASSIGN,BS023=3
*LIMIT,T=30,C=101023,PR=300
*RESTART
  5   6   3   789
  2.6698E-04  56332   54
*END
```
Definitions and Rules (with Overlays)

- SAVEF may be called from an overlay but not a segment.
- If SAVEF is called from an overlay, it must be loaded with the control program.
- If overlays are used with SAVEF, the save tape will have the following format:
  a) First logical file - central memory
  b) End of file mark
  c) Second logical file - overlays and their associated segments
  d) End of file mark
- If overlays and the history file option are used with SAVEF, the save tape will have the following format:
  a) First logical file - central memory
  b) End of file mark
  c) Second logical file - overlays and their associated segments
  d) Third logical file - user-constructed data file
  e) End of file mark
- If only the history file option is used with SAVEF, the save tape will have the following format:
  a) First logical file - central memory
  b) End of file mark
  c) Second logical file - user-constructed data file
  d) End of file mark
- If the history file option is used (i.e., IHISW=1), then the user must furnish his own subroutine called WTHIS. SAVEF will do a return jump to this routine. The routine may perform any operations necessary to move data onto the save tape. The normal return will enable SAVEF to finish.
SAVEF (cont'd)

Definitions and Rules (with Overlays) (cont'd)

. When the history file is used, the following sequence of events will take place at restart time:

a) The first logical file containing central memory will be moved to central memory.

b) If overlays were used, they will be moved to the drum.

c) Control will be returned to the program which called SAVEF at the actual statement which performed the call.

d) The save tape will now be positioned at the beginning of the logical data file previously written by the user's WTHIS subroutine.

e) It is now up to the user's program to move the data file back from the save tape to its original storage position.

f) Computation can now continue as before.

Returns

There are two argument returns from an FARG=SAVEF(ISAV) type of call:

a) The normal return where execution did not cease and calculations are to continue in a normal fashion:

\[ \text{FARG} = 0 \text{ for ISAV} = 0 \text{ or ISAV} = 1 \]

b) The restart return where control is being returned to the calling program after a restart:

\[ \text{FARG} \neq 0 \text{ for ISAV} = 0 \text{ or ISAV} = 1 \]

When programming with SAVEF, it is necessary to have in mind a few facts. The call statements should be placed at the logical termination of a calculation cycle and before another cycle is started. In some instances it is necessary to program special logic for restart purposes, since on a restart the program is started back through the FARG=SAVEF(ISAV) statement. FARG is a floating point argument which is non-zero when the program is being restarted through the use of a restart deck. FARG equals zero upon return from all other calls.
Library Routines - Utility

SAVEF (cont'd)

Error Messages

SW6 TERM BY SAVE
CHECKSUM ERROR
DRUM WRITE OF OVERLAY FAILED
DRUM READ OF OVERLAY FAILED
SAVE TAPE WRITE ERROR
SAVE TAPE READ ERROR

Summary

The following table summarizes sense switch setting and ARG=0 or 1 in using the SAVEFunction:

<table>
<thead>
<tr>
<th>Sense Switch 6</th>
<th>Arg</th>
<th>Arg = 0</th>
<th>Arg = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td></td>
<td>Save and terminate</td>
<td>Save and terminate</td>
</tr>
<tr>
<td>OFF</td>
<td></td>
<td>No action (program continues)</td>
<td>Save (program continues)</td>
</tr>
</tbody>
</table>
SORT

Purpose: Sorts an array A (or K) in ascending order from A(I) to A(J). This routine will sort Hollerith data in alphabetic order. For each column, the order will be: 1) letters A-Z, 2) numbers 0-9, 3) blanks, 4) dashes.

Fortran Function: CALL SORT (A, I, J)

Entry Point: SORT

Ascent Calling Sequence:
- RJ  SOR{T
- EQ  *+4
- CON A
- CON I
- CON J

Normal Return: A is reordered in ascending order

Error Message: None

Storage: 2118 locations

Accuracy: Not applicable

Timing: Depends on the size of the array (15 ms for 200 elements of A)

Method: The array may be fixed or floating point. The ordering is done by integer subtraction. If the array is floating point, the numbers must be in normalized form. The array may be dimensioned for $2^{16}$ elements.
Sorting Routine

Method Arguments:

A The array to be sorted.
I The first element of the array to be sorted.
J The last element of the array to be sorted.

I should be less than J. The array is repeatedly split into segments to create a fast sort.
SSW

To set and test sense lights and sense switches.

To test the sense switch (lite) use:

\[
\begin{align*}
\text{IF (SENSE SWITCH } i) & n_1, n_2 \\
\text{IF (SENSE LITE } i) & n_1, n_2
\end{align*}
\]

To set the sense switch (lite) use:

\[
\begin{align*}
\text{SENSE SWITCH } i \\
\text{SENSE LITE } i
\end{align*}
\]

Entry Points

Ascent Call

The number of the sense switch or lite is sent in XI.
The flag for the result of the test is returned in XI.

Storage

148 locations
# INTRODUCTION

The following is a list of entry points into the PLOT routine described in this section. The routines are divided into the following categories: scaling, plotting, writing, control, backgrounds, and plot instruction manipulation. Information on instruction format is included at the end of this section.

<table>
<thead>
<tr>
<th>Entry</th>
<th>Arguments</th>
<th>Alternate Arguments</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scaling</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>XA, XB, YA, YB, XC, XD, YC,</td>
<td></td>
<td>402</td>
</tr>
<tr>
<td></td>
<td>YD, LTYPEx</td>
<td></td>
<td>403</td>
</tr>
<tr>
<td>PSCALE</td>
<td>SCALEX, SCALEY</td>
<td></td>
<td>405</td>
</tr>
<tr>
<td>PORCN</td>
<td>X, Y</td>
<td>MX, MY</td>
<td>405</td>
</tr>
<tr>
<td>Plotting</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FRSTPT</td>
<td>X, Y</td>
<td>MX, MY</td>
<td>406</td>
</tr>
<tr>
<td>VECTOR</td>
<td>X, Y</td>
<td>MX, MY</td>
<td>407</td>
</tr>
<tr>
<td>POINT</td>
<td>X, Y</td>
<td>MX, MY</td>
<td>407</td>
</tr>
<tr>
<td>PSYM</td>
<td>X, Y, ICHR, ISIZ, ICAS, IP</td>
<td>MX, MY, ICHR, ISIZ, ICAS, IP</td>
<td>407</td>
</tr>
<tr>
<td>LINE</td>
<td>XA, XB, YA, YB</td>
<td>MXA, MYA, MXB, MYB</td>
<td>407</td>
</tr>
<tr>
<td>CURVE</td>
<td>XAD, YAD, N</td>
<td>MXAD, MYAD, N</td>
<td>407</td>
</tr>
<tr>
<td>Writing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PWRT</td>
<td>X, Y, LOC, N, ISIZ, IOR</td>
<td>MX, MY, LOC, N, ISIZ, IOR</td>
<td>410</td>
</tr>
<tr>
<td>Control</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLUSH</td>
<td></td>
<td></td>
<td>415</td>
</tr>
<tr>
<td>FRAME</td>
<td></td>
<td></td>
<td>416</td>
</tr>
<tr>
<td>OPTION</td>
<td>ICAS, INT, ITAL, IOR</td>
<td></td>
<td>417</td>
</tr>
<tr>
<td>DASHLN</td>
<td>IPATRN</td>
<td></td>
<td>418</td>
</tr>
<tr>
<td>MXMY</td>
<td>MX, MY</td>
<td></td>
<td>420</td>
</tr>
<tr>
<td>Backgrounds</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GRID</td>
<td>MGRX, MINRX, MGY, MINRY</td>
<td></td>
<td>421</td>
</tr>
<tr>
<td>PERIM</td>
<td>MGRX, MINRX, MGY, MINRY</td>
<td></td>
<td>422</td>
</tr>
<tr>
<td>PERIML</td>
<td>MGRX, MINRX, MGY, MINRY</td>
<td></td>
<td>423</td>
</tr>
<tr>
<td>GRIDL</td>
<td>MGRX, MINRX, MGY, MINRY</td>
<td></td>
<td>423</td>
</tr>
<tr>
<td>HALFAX</td>
<td>MGRX, MINRX, MGY, MINRY,</td>
<td>MGRX, MINRX, MGY, MINRY,</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td>X, Y, IXLAB, IYLAB</td>
<td>MX, MY, IXLAB, IYLAB</td>
<td></td>
</tr>
<tr>
<td>AXES</td>
<td>X, Y</td>
<td>MX, MY</td>
<td>424</td>
</tr>
<tr>
<td>GRIDALL</td>
<td>MGRX, MINRX, MGY, MINRY</td>
<td></td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>IXLAB, IYLAB, IYGP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TICS</td>
<td>MAJOR, MINOR</td>
<td></td>
<td>425</td>
</tr>
<tr>
<td>TICK4</td>
<td>MGRX, MINRX, MGY, MINRY</td>
<td></td>
<td>426</td>
</tr>
<tr>
<td>LABMOD</td>
<td>IFMTX, IFMTY, NUMX, NUMY, ISIZX,</td>
<td></td>
<td>426</td>
</tr>
<tr>
<td></td>
<td>ISIZY, IXDEC, IYDEC, IXOR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plot Instruction Manipulation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLASH1</td>
<td>IFIELD, MAXLEN</td>
<td></td>
<td>430</td>
</tr>
<tr>
<td>FLASH2</td>
<td>IFORM, LENGTH</td>
<td></td>
<td>430</td>
</tr>
<tr>
<td>FLASH3</td>
<td>IFORM</td>
<td></td>
<td>431</td>
</tr>
<tr>
<td>FLASH4</td>
<td>IFWA, LWA, IFORM</td>
<td></td>
<td>432</td>
</tr>
</tbody>
</table>
The PLOT routine provides capability for graphic output. PLOT is written in machine language and is on the library tape. It currently has 29 entry points, 10 of which have alternate argument lists.

Many (listed on page 400)

FLASH BUFFER OVERFILL
X OR Y COORDINATE 0 OR NEGATIVE WITH LOG SCALING
FRAME LIMIT EXCEEDED
SCALING ARGUMENTS 0 OR NEGATIVE WITH LOG SCALING
TRIED TO START FLASH WHILE IN FLASH
MORE THAN 250000 PLOT INS. - ASSIGN DD80=59

1600

Variable, depending on what is plotted.

The dd80C is a cathode ray tube (CRT) plotter. It is designed to run on-line with the CDC 6600 computer. The plotter consists of a CRT, a 35mm microfilm camera to photograph images projected on the CRT, and a logic control section. A 1024 by 1024 horizontal (X), vertical (Y) coordinate system controls beam positioning (plotting). Displays are of three types: lines, points, and symbols. Symbols are drawn by a character generator in the hardware. They may be of four sizes, two orientations, and two intensities, with the additional option of italics in one direction. There are symbols for the entire alphabet in capitals and lower case, the digits 0 - 9, the Greek alphabet, and a variety of punctuation and special symbols.
SCALING

General Discussion

The scaling routines establish a mapping between the user's Cartesian X,Y plane and the dd80's I,J plan. The I,J plane is defined by 1024 possible addresses in I and J.

The mapping equations used are either

1) \[ i = ax + b \quad j = cy + d \]

or 2) \[ i = a\log{x} + b \quad j = c\log{y} + d \]

The SET routine establishes \(a\), \(b\), \(c\), and \(d\), as well as controlling the linear-log combination. PSCALE defines \(a\) and \(c\) only. PORGN defines \(b\) and \(d\) only. \(a\) and \(c\) are initially set to 1023.; \(b\) and \(d\) are initially set to 0.; and the type of mapping is linear-linear.
SET (XA, XB, YA, YB, XC, XD, YC, YD, LTYPE)

XA, XB, YA, YB
(floating)

Numbers between 0. and 1. defining the portion of the CRT to be used. (0.,1.,0.,1.) specifies the entire area. The following restrictions apply:

0. ≤ XA < XB ≤ 1.
0. ≤ YA < YB ≤ 1.

XC, XD, YC, YD
(floating)

Numbers indicating minimum and maximum values of the array to be plotted in the CRT area specified by arguments 1-4.

XC maps to XA
XD maps to XB
YC maps to YA
YD maps to YB

In either linear or log mapping, the only restrictions are that

XC ≠ XD
YC ≠ YD

that is, XD may be larger or smaller than XC and similarly for YC and YD.

LTYPE
(integer)

Flag indicating log or linear mapping.

<table>
<thead>
<tr>
<th>LTYPE</th>
<th>X direction</th>
<th>Y direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>linear</td>
<td>linear</td>
</tr>
<tr>
<td>2</td>
<td>linear</td>
<td>log</td>
</tr>
<tr>
<td>3</td>
<td>log</td>
<td>linear</td>
</tr>
<tr>
<td>4</td>
<td>log</td>
<td>log</td>
</tr>
</tbody>
</table>

Purpose
To establish a mapping of the type specified by LTYPE between the area of the CRT given by XA, XB, YA, YB and a user's plane with an X range of XC (left) to XD (right) and a Y range of YC (bottom) to YD (top). This scaling will be in effect for all subsequent plotting until changed by another call to SET, PORGN or PSSCALE.
Example

CALL SET (.1,.8,.3,.5,0.,20.,10.,1000.,2)

Linear case (X)

<table>
<thead>
<tr>
<th>MINX</th>
<th>MAXX</th>
<th>RCRT</th>
<th>RCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1 x 10^23 = 102</td>
<td>.8 x 10^23 = 818</td>
<td>818 - 102 = 716</td>
<td>20 - 0 = 20</td>
</tr>
</tbody>
</table>

Range of CRT addresses (RCRT) is 816 - 102 = 716

Range of calling addresses (RCA) is 20 - 0 = 20.

X scale factor (a) then equals \( \frac{716}{20} = 35.8 \)

X zero point address (b) is 102 = b + 35.8x0.

b = 102

d = 239

c = \frac{205}{3} = 68.33

Log case (Y)

<table>
<thead>
<tr>
<th>MINY</th>
<th>MAXY</th>
</tr>
</thead>
<tbody>
<tr>
<td>.3 x 10^23 = 307</td>
<td>.5 x 10^23 = 512</td>
</tr>
</tbody>
</table>

RCRT = 512 - 307 = 25

RCA = \log 10000 - \log 10 = 3.0

d = 239

c = \frac{205}{3} = 68.33

The following figure shows the mapping described in the above example:
PScale (SCALEX,SCALEY)

Purpose: To set directly the X and Y scale factors (a and c) in the mapping equations:

1) \( i = ax + b \) \quad j = cy + d
or 2) \( i = a \log x + b \) \quad j = c \log y + d

Method: \( a = \text{SCALEX} \) \quad \( c = \text{SCALEY} \)

FORGN (X,Y)

Purpose: Coordinates in terms of current scale factors and zero point addresses of desired new zero point addresses. (See page 402, General Discussion)

Method: \( b = X - 1 \) \quad \( d = Y - I \)

FORGN (MX,MY) (alternate entry)

Purpose: New zero point coordinates in range

Method: \( b = MX - 1 \) \quad \( d = MY - 1 \)
The plotting routines enable the user to draw points, lines, and individual symbols. Scaling is as specified by initial conditions or the scaling routines. When LTYPE is other than 1, the appropriate log of the calling number is plotted if floating point coordinates are specified.

**Note** on arguments X,Y and MX,MY:

Wherever the coordinate arguments of a routine are X and Y (floating point), alternate arguments MX and MY (integer) may be used.

The floating point arguments are used to determine a CRT address using the following formulas:

\[
\text{CRT ADDRESS (i)} = a \times X \ (\text{or log}X) + b \\
\text{CRT ADDRESS (j)} = c \times Y \ (\text{or log}Y) + d
\]

The CRT addresses are integers in the range 0 ≤ CRT address ≤ 1023.

The a, b, c, and d are established initially as 1023., 0., 1023., 0., respectively, and can be modified by SET, PORGN, and PSSCALE. (See page 402 for discussion of a, b, c, and d.)

The integer arguments determine the i and j by the following formulas:

\[
\text{CRT ADDRESS (i)} = MX - 1 \\
\text{CRT ADDRESS (j)} = MY - 1
\]

Note that while the range of CRT addresses (i and j) is 0 to 1023, the range of MX and MY is 1 to 1024. This is because it is impossible to distinguish between an integer zero and a floating point zero on the 6600. The routines, therefore, always treat a zero as a floating point number.

**Example:** POINT, for instance, may be called in any of these ways:

\[
\text{CALL POINT (X,Y) CALL POINT (X,MY) CALL POINT (MX,Y) CALL POINT (MX,MY)}
\]
These alternate arguments apply to the X and Y coordinates of the following routines:

- PORGN
- PSYM
- POINT
- FRSTPT
- VECTOR
- PWRT
- LINE
- CURVE
- AXES
- HALFAAX
- GRIDALL

* FRSTPT (X,Y)  
  X,Y (floating)  
  Coordinates of the beginning point of a line.  
  Purpose To establish the base point of a line. No plotting is done by FRSTPT.

* VECTOR (X,Y)  
  X,Y (floating)  
  Coordinates of an end point of a line.  
  Purpose To plot a straight line segment from the (X,Y) coordinates of the immediately previous plotter instruction to the coordinates given by (X,Y). A curve is often drawn with an initial call to FRSTPT followed by as many successive calls to VECTOR as there are points along the line.

* POINT (X,Y)  
  X,Y (floating)  
  Coordinates of point to be plotted.  
  Purpose To plot a point at (X,Y).

*See Note, page 406.
* PSYM (X,Y,ICHR,ISIZ,ICAS,IP)

X,Y (floating) Coordinates of point.
ICHR (Hollerith) Character to be plotted.
ISIZ (integer) Size of character (see Table 1, page 411)
ICAS (integer) 0 = upper; 1 = lower case
IP (integer) 1 = symbol only; 2 = symbol plus line to symbol from most recent plot point.

Purpose
To draw the symbol specified by ICHR and ICAS of size ISIZ at (X,Y) connected by a line to previous plot point if IP = 2.
If an octal number if used instead of a IH (character), it should be left-adjusted within the word. Intensity, orientation and italics may be controlled by a prior call to OPTION.

* LINE (XA,YA,XB,YB)

XA,YA (floating) Coordinates of beginning point.
XB,YB (floating) Coordinates of end point

Purpose
To draw a line from (XA,YA) to (XB,YB).

* CURVE (XAD,YAD,N)

XAD (floating) An array of X coordinates.
XAD (floating) An array of Y coordinates.
N (integer) The number of points on the curve.

Purpose
To draw a series of line segments connecting the N points specified in the arguments. The routine assumes the X and Y arrays are ordered. If integers are in the array, they will be plotted without scaling.

*See Note, page 406.
Example

*FORTRAN

PROGRAM CIRCLE
DIMENSION X(100), Y(100)
CALL GRID(1, 1, 1, 1)
PI=3.1416
R=3.
D=PI/36
DO 1 I=1, 73
X(I)=R*COS(D*I)
Y(I)=R*SIN(D*I)
CALL PORGN(300, 400)
CALL PSSCALE(70, 50)
CALL CURVE(X, Y, 73)
CALL POINT(0, 0)
CALL PORGN(600, 700)
CALL PSSCALE(100, 100)
CALL FRSTPT(X(1), Y(1))
DO 2 I=2, 73
2 CALL VECTOR(X(I), Y(I))
CALL LINE(600, 700, X(I), Y(I))
CALL PORGN(100, 900)
CALL PSSCALE(30, 30)
CALL PSYM(0, 900, 1HP, 3, 0, 1)
CALL CURVE(X, Y, 73)
CALL FRAME
END
* IWRT (X, Y, LOC, N, ISIZ, IOR)

X, Y (floating) Coordinates of the center of the first character to be written.
LOC (integer) The display code (DPC) array to be written.
N (integer) The number of characters to be written. LOC should be dimensioned appropriately if more than 10 characters are to be written.
ISIZ (integer) Character size (see Table 1, page 411).
IOR (integer) Character (and line) orientation (see Table 2, page 411).

There are three other variables which may be preset that affect the tabular output: intensity (INT), case (ICAS), and italics (ITAL). (See Table 2, page 411.) These variables are all set by a call to OPTION. Note also that ICAS can be set by a call to PSYM. Whenever one of these variables is set, it then remains unchanged until modified by the calling program.

Purpose

PWRT, the writing routine, permits writing Hollerith information on the dd80. 118 characters are available. These may be written in any of four sizes, two orientations, two intensities, and in either block print or italics.

*See Note, page 406.
Table 1. Character Sizes

<table>
<thead>
<tr>
<th>ISIZ</th>
<th>Maximum Char/Line</th>
<th>Maximum Lines/Frame</th>
<th>Increment Along Line</th>
<th>Increment Between Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>128</td>
<td>64</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>86</td>
<td>43</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>22</td>
<td>24</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 2. Character Controls

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOR</td>
<td>writes in +X direction</td>
<td>writes in +Y direction</td>
</tr>
<tr>
<td>INT</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>ICASE</td>
<td>upper</td>
<td>lower</td>
</tr>
<tr>
<td>ITAL</td>
<td>block print</td>
<td>italics - permissible</td>
</tr>
<tr>
<td></td>
<td>only with IOR = 0</td>
<td>only with IOR = 0</td>
</tr>
</tbody>
</table>

Table 3. PWRT Internal Flag Table

<table>
<thead>
<tr>
<th>Tab Bit</th>
<th>Lower Case</th>
<th>Carriage Return</th>
<th>Italics</th>
<th>Not Used</th>
<th>High Intensity</th>
<th>Octal Value</th>
<th>Hollerith (PWRT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>= 00</td>
<td>= 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>= 01</td>
<td>= 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>= 04</td>
<td>= 4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 05</td>
<td>= 5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>= 10</td>
<td>= 8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 11</td>
<td>= 9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 14</td>
<td>= $4</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 15</td>
<td>= $5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>= 20</td>
<td>= +</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>= 21</td>
<td>= A</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>= 24</td>
<td>= D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 25</td>
<td>= E</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>= 30</td>
<td>= H</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>= 31</td>
<td>= I</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>= 34</td>
<td>= )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>= 35</td>
<td>= $E</td>
</tr>
</tbody>
</table>

Bit = 1 means YES; Bit = 0 means NO.
Library Routines - PLOT

Table 4. dd80 Character Set

<table>
<thead>
<tr>
<th>dd80 Characters ICAS=0</th>
<th>Keypunch Characters</th>
<th>Display Code</th>
<th>dd80 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>00</td>
<td>-</td>
</tr>
<tr>
<td>A</td>
<td>a</td>
<td>01</td>
<td>21</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>02</td>
<td>22</td>
</tr>
<tr>
<td>C</td>
<td>c</td>
<td>03</td>
<td>23</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>04</td>
<td>24</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>05</td>
<td>25</td>
</tr>
<tr>
<td>F</td>
<td>f</td>
<td>06</td>
<td>26</td>
</tr>
<tr>
<td>G</td>
<td>g</td>
<td>07</td>
<td>27</td>
</tr>
<tr>
<td>H</td>
<td>h</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>I</td>
<td>i</td>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>J</td>
<td>j</td>
<td>12</td>
<td>41</td>
</tr>
<tr>
<td>K</td>
<td>k</td>
<td>13</td>
<td>42</td>
</tr>
<tr>
<td>L</td>
<td>l</td>
<td>14</td>
<td>43</td>
</tr>
<tr>
<td>M</td>
<td>m</td>
<td>15</td>
<td>44</td>
</tr>
<tr>
<td>N</td>
<td>n</td>
<td>16</td>
<td>45</td>
</tr>
<tr>
<td>O</td>
<td>o</td>
<td>17</td>
<td>46</td>
</tr>
<tr>
<td>P</td>
<td>p</td>
<td>20</td>
<td>47</td>
</tr>
<tr>
<td>Q</td>
<td>q</td>
<td>21</td>
<td>50</td>
</tr>
<tr>
<td>R</td>
<td>r</td>
<td>22</td>
<td>51</td>
</tr>
<tr>
<td>S</td>
<td>s</td>
<td>23</td>
<td>62</td>
</tr>
<tr>
<td>T</td>
<td>t</td>
<td>24</td>
<td>63</td>
</tr>
<tr>
<td>U</td>
<td>u</td>
<td>25</td>
<td>64</td>
</tr>
<tr>
<td>V</td>
<td>v</td>
<td>26</td>
<td>65</td>
</tr>
<tr>
<td>W</td>
<td>w</td>
<td>27</td>
<td>66</td>
</tr>
<tr>
<td>X</td>
<td>x</td>
<td>30</td>
<td>67</td>
</tr>
<tr>
<td>Y</td>
<td>y</td>
<td>31</td>
<td>70</td>
</tr>
<tr>
<td>Z</td>
<td>z</td>
<td>32</td>
<td>71</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>33</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>34</td>
<td>01</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>35</td>
<td>02</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>36</td>
<td>03</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>37</td>
<td>04</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>40</td>
<td>05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dd80 Characters ICAS=0</th>
<th>Keypunch Characters</th>
<th>Display Code</th>
<th>dd80 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>r</td>
<td>41</td>
<td>06</td>
</tr>
<tr>
<td>7</td>
<td>s</td>
<td>42</td>
<td>07</td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td>43</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>y</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>+</td>
<td>blank</td>
<td>45</td>
<td>20</td>
</tr>
<tr>
<td>-</td>
<td>c</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td>#</td>
<td>t</td>
<td>47</td>
<td>54</td>
</tr>
<tr>
<td>/</td>
<td>;</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>(</td>
<td>blank</td>
<td>51</td>
<td>74</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>,</td>
<td>de</td>
<td>,</td>
<td>de</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dd80 Characters ICAS=0</th>
<th>Keypunch Characters</th>
<th>Display Code</th>
<th>dd80 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>r</td>
<td>41</td>
<td>06</td>
</tr>
<tr>
<td>7</td>
<td>s</td>
<td>42</td>
<td>07</td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td>43</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>y</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>+</td>
<td>blank</td>
<td>45</td>
<td>20</td>
</tr>
<tr>
<td>-</td>
<td>c</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td>#</td>
<td>t</td>
<td>47</td>
<td>54</td>
</tr>
<tr>
<td>/</td>
<td>;</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>(</td>
<td>blank</td>
<td>51</td>
<td>74</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>,</td>
<td>de</td>
<td>,</td>
<td>de</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dd80 Characters ICAS=0</th>
<th>Keypunch Characters</th>
<th>Display Code</th>
<th>dd80 Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>r</td>
<td>41</td>
<td>06</td>
</tr>
<tr>
<td>7</td>
<td>s</td>
<td>42</td>
<td>07</td>
</tr>
<tr>
<td>8</td>
<td>x</td>
<td>43</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>y</td>
<td>44</td>
<td>11</td>
</tr>
<tr>
<td>+</td>
<td>blank</td>
<td>45</td>
<td>20</td>
</tr>
<tr>
<td>-</td>
<td>c</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td>#</td>
<td>t</td>
<td>47</td>
<td>54</td>
</tr>
<tr>
<td>/</td>
<td>;</td>
<td>50</td>
<td>61</td>
</tr>
<tr>
<td>(</td>
<td>blank</td>
<td>51</td>
<td>74</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>52</td>
<td>34</td>
</tr>
<tr>
<td>,</td>
<td>de</td>
<td>,</td>
<td>de</td>
</tr>
</tbody>
</table>

Writing
Internal Flags

It is possible to modify intensity, case, and italics at any point while writing the array. In addition, the writing can be forced to start a new line at any time (the new line starts directly under the start of the previous line). This is accomplished by signaling the dd80 with an octal 77 that the next six bits are function control bits. The apostrophe on the keypunch is interpreted as a 778. The proper six bits to follow the 778 may be selected from Table 3, page 411.

If the tab bit is on the arithmetic sum of the lower five bits (from 0 to 3110) is multiplied by 32 and the resulting number used as the new coordinate along the line (X in 0 orientation; Y in 1 orientation). When the tab bit is used, the case, carriage return, italics, and intensity are not changed.

Example: To write

This is the way to change intensity, style and case?

the DPC array to PWRT should be

T'+HIS IS THE '4WAY'+ TO 'OCHANGE '5INTENSITY, S'ETYLE and '+CASE 'D$S
Note on LOC

LOC is the display code (DPC) array to be written. Each character is defined as a six-bit binary constant. The dd80 also uses a six-bit constant for each character, but an entirely different code from the DPC. There are only 48 legal DPC characters (convertible from keypunch data) and 63 legal dd80 codes. To make the full dd80 set available, a $ convention has been added. For instance, to get a $ (ICAS = 0) on the dd80, the 2 characters $P$ are used. The $ itself is not written on the dd80 but causes 108 to be added to the dd80 code for P(47), giving a 57, the dd80 code for $. The price of this system is that the $ itself is never available as a character during PWRT. (It is, however, available as a single character in PSYM.)

If LOC is defined as an octal constant(s) in the program instead of a Hollerith set, the $ could be directly expressed as a 748.

When calculating the number of characters (argument N), all flags and $ must be included, even though they do not appear in the final output.

The four easiest ways to generate LOC are

1) Read it in directly as data in an A format specification.
2) Define it arithmetically or in a DATA statement as a Hollerith or octal constant.
3) Encode it. For example, if you make the following calculation and want to write it on the dd80, use the following sequence:

   DIMENSION LTIT(3)
   Z = SINF(A)
   ENCODE (26,100,LTIT)A,Z
   100 FORMAT (10HTHE SIN $F,F6.2,3H is,$F7.3)
   CALL PWRT (X,Y,LTIT,26,ISIZ,I0R)

4) Use it directly as an argument in PWRT, e.g.,

   CALL PWRT(X,MY,19HTHIS IS AN ARGUMENT,19,2,0)
For purposes of overall execution speed, plotter instructions are not sent directly to the dd80 when formed by a call to one of the routines in PLOT. Normally up to 425 instructions are collected in an array, or buffer, internal to PLOT and then sent all together to the dd80, or more precisely to the system output file, when the buffer is full.

There are two potential pitfalls in this scheme. First, when the program terminates execution, there may still be instructions in the buffer which will not reach the output file (dd80) unless a specific request is made to send them there. FLUSH is a routine to do this. The buffer is also cleared by each call to FRAME (which internally calls FLUSH after generating the frame advance instruction). It follows then that any program using the PLOT package should have as its last plotting routine call a call to either FLUSH or FRAME.

The second potential pitfall is when dd80 instructions generated by the PLOT routines are intermixed with dd80 instructions generated by WRITE or PRINT instructions when printer output is assigned to the dd80. In this case the dd80 instructions are being sent to the output file via two quite different routines, and it is easy (perhaps almost inevitable) to cause the instruction sequence in the output file to be different from the sequence in which the instructions were originally generated. To enforce the same order, a FLUSH (or FRAME) call should precede each change from PLOT type instructions to PRINT or WRITE type instructions.
Plot Buffer Control (cont'd)

Care should be additionally exercised in controlling whether the printing is on the same frame as the plots. Printer output sent to the dd80 follows the following rules:

1) If the last dd80 instruction in the output file (not the PLOT buffer) was from PLOT or if there have been no previous printer lines, assume the top of the page (line one).

2) Issue a frame advance only if
   a) the carriage control character requests it (IHI)
   b) the line count reaches 64.

The PLOT buffer may be bypassed altogether by use of the FLASH routines. These routines cause the plotter instructions to be accumulated in a user-designated array and only sent to the output file when so requested by the user (FLASH3). See page 431 for details.

FLUSH (no arguments)

Purpose To send all calculated but unexecuted plotter instructions to the dd80. PLOT has a 255-word instruction buffer which holds 425 plotter instructions. This is normally dumped to the output file (or dd80) only when full. FLUSH forces this buffer to the output file even though it is not full. FLUSH (or FRAME) must be called at the end of plotting in any program to make sure all calculated instructions actually reach the dd80. It should also be called before switching from graphic output to printed output assigned to the dd80 by an *ASSIGN card.
FRAME (no arguments)

Purpose To advance the film one frame. FRAME also forces a PLOT buffer flush.

Other Control Routines The following four routines do not actually plot, but give control of some special capabilities of the hardware and software.

OPTION (ICAS, INT, ITAL, IOR)

ICAS (integer) 0 = upper case 1 = lower case
INT (integer) 0 = low intensity 1 = high intensity
ITAL (integer) 0 = block print 1 = italics
IOR (integer) 0 = +X direction (horizontal) 1 = +Y direction (vertical)

Purpose To set flags for the options above. ICAS, ITAL and IOR apply only to symbols and test (PSYM and PWRT). INT applies to all plotting. The normal value of all these flags is 0 (to which they are set initially). Once set, each flag stays set until changed by another OPTION call or a call to a routine using the flag as an argument. Note that GRID, PERIM, GRIDL, PERIML, GRIDALL and HALFAX leave the dd80 in high intensity.
DASHLN (IPATRN)

**IPATRN (integer)**

A 10-bit octal constant $0 \leq \text{IPATRN} \leq 1777_8$ defining the dash pattern for vectors.

**Purpose**

To make a variety of dashed line types available. IPATRN controls the blank-unblank circuitry in the dd80. A 1 bit is an unblank; a 0 bit, a blank.

The pattern set by DASHLN stays in force for all plotting until reset by another DASHLN call or a call to PWRT (which sets the pattern back to $1777_8$). For example,

- A bit pattern of $1777_8 (=1111111111_2)$ makes a solid line.
- A bit pattern of $1430_8 (=1100011000_2)$ makes a line $2/10$ solid, $3/10$ blank, $2/10$ solid, $3/10$ blank.

The DASHLN feature is based on the timing clock of the dd80 and different line lengths are a function both of the number of clock cycles and the beam speed. Therefore the individual dashes from a given DASHLN pattern will not necessarily be the same length in line segments of different lengths.
DASHLN (cont'd)

Example

```
PROGRAM TEST
DIMENSION IDASH(10), IL(16), IB(16), IC(16), ID(4, 7)
DATA (IDASH=1774B, 1760B, 1700B, 1400B, 1430B, 1736B, 1766B)
DATA (ID=40H1111111100 B'AINARY OR '11774 O'ACTAL',
2 40H111110000 B'AINARY OR '111760 O'ACTAL',
3 40H111100000 B'AINARY OR '111700 O'ACTAL',
4 40H110000000 B'AINARY OR '111400 O'ACTAL',
5 40H110011000 B'AINARY OR '111430 O'ACTAL',
6 40H1111011110 B'AINARY OR '111736 O'ACTAL',
7 40H1111101110 B'AINARY OR '111766 O'ACTAL'
)
DATA (IB=1*1*1*1*40B*100B*140B)
DATA (IC=2*4*16*8*0*0*0)
DATA (IL=600B, 36UB, 70B*16B*8*4*2)
CALL OPTION(0*1*0*0)
DO 2 J=1, 7
K=600-B0*J
CALL PWRT(1*K+60B, ID(J), 40, 2*0)
CALL DASHLN[IDASH(J)]
DO 1 I=1, 7
1 CALL LINE(IB(I)+K+IC(I)+IB(I)+IL(I)+K+IC(I))
2 CONTINUE
CALL PWRT(1*600*30*20, 'AASHED', 'IL'AINE', 'IP'PAATERN'S+30*3*0)
CALL FRAME
END
```

Dashed Line Patterns

11111111100 Binary or 1774 Octal

11111110000 Binary or 1760 Octal

11111000000 Binary or 1700 Octal

11000000000 Binary or 1400 Octal

11000110000 Binary or 1430 Octal

11110111110 Binary or 1736 Octal

11111011110 Binary or 1766 Octal
Library Routines - PLOT

Control

MXMY (MX,MY)  MX,MY (integer)  Names of 2 core cells in the program in which the most recent CRT coordinates generated by any plot call will be placed.

Purpose  To return to the calling program the integer coordinates of the last plotter instruction. This is useful in some kinds of titling.
BACKGROUNDs

General Discussion

These routines are convenience routines for drawing graph paper, axes, and other odds and ends.

All the routines use the combination of linear-log scaling specified in SET (linear-linear if SET has not been called).

All backgrounds—or graph paper—are drawn over the area specified by the first four arguments of SET; and if automatically annotated, the annotation assumes the mapping specified by arguments 5-8 of SET. See figure on page 404. Several of them (GRIDALL, GRID, GRIDL, PERIM, PERIML, HALFAX) have arguments MGRX,MINRX,MGRY,MINRY. These arguments control the number of major and minor divisions in the graph paper or perimeter. Major division lines, or tick marks, are drawn in high intensity; minor lines or ticks, in low intensity. The number of divisions refers to the holes between lines rather than the lines themselves. This means that there is always one more major division line than the number of major divisions, and there is one less minor division line than minor divisions (per major division).
The meaning of the argument in the log case is completely different from the meaning in the linear case, as follows:

<table>
<thead>
<tr>
<th>Linear</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MGRX</strong> (integer)</td>
<td>The number of major divisions along the X-axis. Divisions are the holes, not the lines.</td>
</tr>
<tr>
<td><strong>MINRX</strong> (integer)</td>
<td>The number of minor divisions per major division. Again, this is the number of holes, not lines.</td>
</tr>
<tr>
<td><strong>MGRY</strong> (integer)</td>
<td>Same as MGRX except for Y-axis.</td>
</tr>
<tr>
<td><strong>MINRY</strong> (integer)</td>
<td>Same as above.</td>
</tr>
</tbody>
</table>

**GRID** *(MGRX,MINRX,MGRY,MINRY)*

- **MGRX,MINRX** (integer) | See above. |
- **MGRY,MINRY** (integer) | |

**Purpose**

To draw graph paper. This routine will draw graph lines in the portion of the CRT specified by a SET call, with the number of major and minor divisions specified by the arguments. Lines at major divisions are high intensity; minor ones, low intensity. On log type axes, the eight subcycle divisions are not marked if MGRX (or MGRY) is other than 1.
PERIM (MGRX, MINRX, MGRY, MINRY)

Arguments same as GRID.

Purpose Same as GRID but interior lines are replaced with tick marks along the edges.

PERIML (MGRX, MINRX, MGRY, MINRY)

Arguments same as GRID.

Purpose Same as PERIM, but each major division is marked with its numerical value.

GRIDL (MGRX, MINRX, MGRY, MINRY)

Arguments same as GRID.

Purpose Same as GRID, but each major division is marked with its numerical value.
HALFAX (MGRX, MINRX, MGRY, MINRY, X, Y, IXLAB, IYLAB)

MGRX, MINRX  (integer)
MGRY, MINRY  (integer)

X, Y  (floating or integer)
The intersection point of the X and Y axes. If floating point, the coordinate is scaled according to SET. If integer, it is assumed to be an absolute CRT coordinate.

IXLAB, IYLAB  (integer)
Flags for line-value labels:

- IXLAB = 0  No X labels
- IXLAB = 1  X-axis labels
- IYLAB = 0  No Y labels
- IYLAB = 1  Y-axis labels

Purpose
To produce orthogonal axes intersecting at X, Y with division tick marks according to MGRX, MINRX, MGRY, and MINRY. These can be labeled if desired.

AXES (X, Y)  X, Y  (floating)
Coordinates of axis intersection.

Purpose
To draw a set of perpendicular axes through point (X, Y) to the minimum and maximum points specified by the first four arguments of SET. If SET has not been called, the axes will be drawn to the edges of CRT.
GRIDALL (MGRX, MINRX, MGRY, MINRY, IXLAB, IYLAB, IHGP)

MGRX, MINRX
MGRY, MINRY
(integer)

Same as GRID.

IXLAB, IYLAB
(integer)

Flags for line value labels along axes:

<table>
<thead>
<tr>
<th>IXLAB</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No X-axis labels</td>
</tr>
<tr>
<td>1</td>
<td>X-axis labels</td>
</tr>
<tr>
<td>0</td>
<td>No Y-axis labels</td>
</tr>
<tr>
<td>1</td>
<td>Y-axis labels</td>
</tr>
</tbody>
</table>

IHGP
(integer)

Flag for background type:

<table>
<thead>
<tr>
<th>IHGP</th>
<th>X-axis Background</th>
<th>Y-axis Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GRID</td>
<td>GRID</td>
</tr>
<tr>
<td>1</td>
<td>GRID</td>
<td>PERIM</td>
</tr>
<tr>
<td>2</td>
<td>GRID</td>
<td>HALFWAX</td>
</tr>
<tr>
<td>4</td>
<td>PERIM</td>
<td>GRID</td>
</tr>
<tr>
<td>5</td>
<td>PERIM</td>
<td>PERIM</td>
</tr>
<tr>
<td>6</td>
<td>PERIM</td>
<td>HALFWAX</td>
</tr>
<tr>
<td>8</td>
<td>HALFWAX</td>
<td>GRID</td>
</tr>
<tr>
<td>9</td>
<td>HALFWAX</td>
<td>PERIM</td>
</tr>
<tr>
<td>10</td>
<td>HALFWAX</td>
<td>HALFWAX</td>
</tr>
</tbody>
</table>

If IHGP = 10, the axis intersection will be taken at the lower left corner of the area specified by SET.

Purpose

To have a general entry point for all backgrounds with option of line labeling on each axis.

TICKS (MAJOR, MINOR)

MAJOR
(integer)

CRT length of major division tick marks. Positive will produce inward pointing ones; negative will produce outward pointing ones.

MINOR
(integer)

Same for minor division tick marks.

Purpose

To allow program control of tick mark length in PERIM, PERIML, GRIDALL, and HALFWAX. MAJOR is initially set to 12 and MINOR is initially set to 8.
TICK4 (MGRX, MINRX, MGRY, MINRY)

- **MGRX** (integer): Length in CRT units of major division tick marks on the X-axis.
- **MINRX** (integer): Length in CRT units of minor division tick marks on the X-axis.
- **MGRY, MINRY** (integer): Same as above for the Y-axis.

**Purpose**

To allow program control of tick mark length in PERIM, PERIML, GRIDALL, and HALFAX. TICK4 allows separate control of X- and Y-axes tick marks. MGRX and MGRY are initially set to 12, and MINRX and MINRY are initially set to 8.

LABMOD (IFMTX, IFMTY, NUMX, NUMY, ISIZX, ISIZY, IXDEC, IYDEC, IXOR)

- **IFMTX** (integer): A Hollerith format specification for X-axis numerical labels in GRIDL, PERIML, GRIDALL, or HALFAX. The specification must start with a left parenthesis and end with a right parenthesis, and should not use more than 10 characters. Only F and E conversions are permissible, such as IFMTX=6H(F8.2).
- **IFMTY** (integer): Same as above for Y-axis labels, such as IFMTY=7H(E10.0).
- **NUMX, NUMY** (integer): The number of characters specified by IFMTX and IFMTY respectively. These would be NUMX=8, NUMY=10 for the above example.
- **ISIZX, ISIZY** (integer): The character size codes for the X and Y labels respectively. Codes are the same as for PWRT and PSYM.

<table>
<thead>
<tr>
<th>ISIZX (or ISIZY)</th>
<th>CRT units/character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
</tbody>
</table>
The decrement in CRT units from the MINX position (specified by the first argument of SET) to the starting X-address of the label specified by IFMTY, NUMY and ISIZY. For example, if the first argument of SET is .1, MINX is 102 (.1*1024). If IXDEC is 60, the label will start at 62 (102-60). To save ordinary condition calculations, the following conventions are used:

If IXDEC = 0, it is set to properly position the Y-axis labels to the left of the Y-axis. The plot package calculates it by:

\[ \text{decrement} = (\text{NUMY} + \frac{1}{2}) \times \text{character size Y} \]

Thus if NUMY = 7 and ISIZY = 2 and IXDEC is set to zero, the X decrement becomes \((7+\frac{1}{2}) \times 16 = 120\).

If IXDEC = 1, it is reset to produce labels to the right of the right Y-axis. The plot package calculates it as follows:

\[ \text{X decrement} = -(\text{MAXX-MINX} + \text{char size Y}) \]

Thus if the first two arguments of SET are .1 and .8, ISIZY = 3 and IXDEC = 1, the X decrement is \(-((.1 \times 1024) - (.8 \times 1024) + 24) = -741\).

The decrements in CRT units from the MINY position (specified by the third argument of SET) to the starting Y-address of the label specified by IFMX, NUMX and ISIZX. For example, if the third argument of SET is .2, MINY is .2*1024 = 205. If IYDEC = 30, the label will start at 205-30 = 175. To save ordinary condition calculations, there are two conventions:

If IYDEC = 0, the decrement is set to produce labels along the bottom. The calculations depend on whether the X-labels are in normal or 90 orientation (see IXOR). For IYDEC = 0, IXOR = 0, the decrement equals the character size; e.g., if ISIZX = 2, the decrement equals 16. For IYDEC = 0, IXOR = 1:

\[ \text{Y decrement} = (\text{NUMX} + \frac{1}{2}) \times \text{character size X} \]

If IYDEC = 1 and IXOR = 0 or 1, the labels will go along the top of the graph. The plot package makes the following calculations:

\[ \text{Y decrement} = -(\text{MAXY-MINY} + \text{char size X}) \]
LABMOD (cont'd)

---

**IXOR**

(integer)

Orientation of the X-axis labels.

- IXOR = 0 normal or +X
- IXOR = 1 90° or +Y

In normal orientation the actual number of digits that are non-blank are centered under the line or tick to which they apply.

**Purpose**

To enable more flexibility in graph paper generation. Note that LABMOD itself does no plotting. It only presets parameters for the graphing routines. LABMOD, if called, must be called after the SET call, as it needs the information derived by SET. It obviously must be called before the background routines, such as PERIML, for which it is presetting parameters.
Example 1

PROGRAM GRAPHS
CALL SET(0., 0., 1., 0., 1., 0., 1., 1.)
CALL GRDIN(11111)
CALL SET(.6, .9, .5, .95, 1., 10000., 3., 3., 05, 3)
CALL GRDIN(111113)
CALL SET(.6, .97, .07, .45, 0., .0., 1.6, 100, 2)
CALL TICKS(-15., -7)
CALL PERIM(40., 60., 1)
CALL SET(.1, .5, .1, .4, 1., -1., 1., 0., 1, .5, 57, 2)
CALL LABM00(mH(F5.2), 7H(F9.7), 7H(F9.1), 5, 3, 1, 6, 0, 0, 0)
CALL TICKS(.0, 15)
CALL PERIML(.4, 2, 10, 1)
CALL SET(.02, .45, .95, 0., 10., 1000., 0., 1)
CALL LABM00(mH(F2.0), 4H(F4.0), 8H(F4.0), 2., 4, 1, 2, 1, 1, 1)
CALL PERIML(.7, 2, 10, 4)
CALL PWRT(270., 40., 6H(PERMIL), 4, 2, 0)
CALL PWRT(100., 40., 5H(PERMIL), 5, 3, 0)
CALL PWRT(270., 490., 5H(PERMIL), 5, 0, 0)
CALL PWRT(800., 490., 4H(PERMIL), 4, 1, 0)
END

\[
\begin{array}{c}
\begin{array}{c}
\text{PERMIL} \\
\end{array}
\end{array}
\]
Example 2

Examples of backgrounds generated by GRIDALL:

```
PROGRAM SAMP
DIMENSION L(3),IHGP(3,3)
DATA(XA=.1),(YA=.05),(R=25),(S=.05)
DATA(L=5HGRID,5HPERIM,6HHALFAX),(IHGP=2,6,10,15,9,0,4,9,8)
IX=1
DO 1 I=1,3
  IY=(YA+(I-1)*(R*S)+.5*R)*1024.
  CALL PWRT(IXIYL(4-I),6,2,0)
  IY=975
  DO 2 I=1,3
    IX=(XA+(I-1)*(R+S)+.5*R)*1024.-40.
    CALL PWRT(IXIY,L(I),6,2,0)
  DO 3 I=1,3
    X1=XA+(I-1)*(R+S) $ X2=X1*R $ IX=512.*(X1+X2)-48.
    DO 3 J=1,3
      Y1=YA+(J-1)*(R+S) $ Y2=Y1+R $ IY=512.*(Y1+Y2)
      CALL SET(X1,X2,Y1,Y2,0.11.,0.1,1)
      CALL GRIDALL(5,3,6,4,00G,IHGP(I,J))
      ENCODE(7,100,M)IHGP(I,J)
100 FORMAT(5HIHGP=,I2)
  3 CALL PWRT(IXIYM,12,0)
  CALL FRAME
END
```

```
GRID

PERIM

HALFAX

GRID

IHGP= A

IHGP= B

IHGP= 9

IHGP= 6

IHGP= 10
```
The FLASH routines (FLASH1, FLASH2, FLASH3, and FLASH4) enable plotting on demand repetitive data, such as complicated grids, without recalculating the plotter instructions each time. This is particularly useful for movie applications. Up to 11 separate blocks of instruction may be held in one program. Blocks may also be regenerated at any time during the program. See discussion of normal instruction handling under Plot Buffer Control, page 415.

FLASH1 (IFIELD, MAXLEN)

IFIELD (integer)  
The name of an array in the program that is to receive completed plotter instructions.

MAXLEN (integer)  
The maximum number of words in IFIELD that may be used (normally the dimension of IFIELD).

Purpose

To initiate storage of plotter instructions in the array IFIELD instead of sending them to the dd80.
FLASH2 (IFORM, LENGTH)

IFORM (integer)  An integer between 0 and 10 designating which block of instructions the FLASH call has generated; i.e., a label which will be used by subsequent FLASH3 calls.

LENGTH (integer)  The actual number of words in IFIELD that have been filled. This is returned in LENGTH.

Purpose  To terminate the process started by FLASH1. After a FLASH2 call, plot instructions again go out to the dd80 normally. All instructions between the FLASH1 and FLASH2 calls are stored in IFIELD.

FLASH3 (IFORM)

IFORM (integer)  An integer between 0 and 10 that was used in FLASH2 for the particular block of instructions to be plotted.

Purpose  To send to the dd80 the complete set of instructions generated between a particular set of FLASH1 and FLASH2 calls.
FLASH4 (IFWA, LWA, IFORM)

IFWA (integer)
The first word of a block of plotter instructions not generated by a FLASH1 and FLASH2 sequence in the program being run.

LWA (integer)
The last word of the block. The total length must be a multiple of 3, because of the method of plotter instruction packing.

IFORM (integer)
An integer between 0 and 10 designating which block of instructions this is; to be used in FLASH3 calls.

Purpose
To establish necessary parameters for FLASH3 use for instructions not generated by FLASH1 and FLASH2 in the program being run.
Program Movie

Example of FLASH Routine Use

DIMENSION IA(10000), IB(10000), etc.

NN = 0
CALL SET (appropriate arguments)

1 CALL FLASH1 (IA, 10000)
   CALL CONTIN (a fictitious continental outline program)
   CALL FLASH2(1, NN)
   WRITE(6, 100) NN

100 FORMAT (1HO, *NUMBER OF WORDS USED FOR CONT OUTLINE = *15)

2 WRITE(7)(IA(K), K = 1, NN) (saving of continental outline instructions on tape for future runs)

DO 20 J = 1, 1000
   CALL DATA(J) (a fictitious data generation routine)
   CALL FLASH1(IB, 10000)
   CALL CONTOUR (a fictitious contouring routine)
   CALL FLASH2(2, NN)
   WRITE(6, 101) NN

101 FORMAT (1HO, *NUMBER OF WORDS USED FOR CONTOUR = *15)

DO 10 I = 1, 3
   CALL FLASH3(1) (continental outline)
   CALL FLASH3(2) (contour)
   CALL LABEL(J) (direct, fictitious titling subroutine)

10 CALL FRAME (direct, contains buffer flush)

20 CONTINUE

END
Example of FLASH Routine Use

The sample program on page 434 would generate the plotter instructions once for the continental outline in IA. Then for each set of new data, it would generate the plotter instructions for the contour in IB. Then three identical frames would be made: the continental outline coming from IA, the contour from IB, and the label and frame advance directly. In this program of 3000 frames, the continental outline is calculated only once, and 1000 instead of 3000 contour maps are generated. On subsequent runs of this program, statements 1 through 2 could be replaced with

1 READ(5,100)NN
100 FORMAT(I5)
    READ 7(IA(K),K=1,NN)
2 CALL FLASH4 (IA(1),IA(NN),1)

This would eliminate recalculating all the plotter instructions for the continental outline.
INSTRUCTION FORMAT

General Discussion

The dd80 receives a 36-bit instruction word from the computer. These are of two formats: normal mode and tabular mode. The particular function to be performed is defined by a 3-bit operation code; and the location, by 10-bit X- and Y-coordinates. The bit arrangement is as below.

<table>
<thead>
<tr>
<th>Orientation</th>
<th>Italic</th>
<th>Point</th>
<th>Intensity</th>
<th>Operation code</th>
<th>Size</th>
<th>Case</th>
<th>X Coordinate</th>
<th>Y Coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>35</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30 29 28 27 26 25 24 23 22 21 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Normal Format

Operation codes 0 - 5 are defined as follows:

<table>
<thead>
<tr>
<th>Operation Code</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Symbol plot. In bits 30-35 is the symbol code, instead of orientation, italics and intensity, which may be preset by an operation code 1 call.</td>
</tr>
<tr>
<td>1</td>
<td>Sets up a vector origin at X,Y, or with bit 31 set (1) plots a point at X,Y. This operation code may also be used to preset orientation, italics and intensity without doing any actual plotting.</td>
</tr>
<tr>
<td>2</td>
<td>Draws a vector from the previous X,Y position to the new one.</td>
</tr>
<tr>
<td>3</td>
<td>Initiates tabular mode for generating text. This sets up the tabular mode options and starting X,Y coordinates. The text, in a different format, follows this initial tabular mode instruction.</td>
</tr>
<tr>
<td>Normal Format (cont'd)</td>
<td>Operation Code</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

Tabular Format

The tabular word format is used for text following an operation code of 3. In this format 6-bit symbol codes are packed 6 to an instruction until a 768 symbol code signals the end of text. At this point the dd80 reverts to normal word mode for the next instruction words.
OTHER USEFUL ROUTINES

The following routines are written in Fortran and are not on the library tape. Decks may be secured from David Robertson.

CALCNT

CALL CALCNT(AM,MX,NY,FLO,HI,FINC,NSET,NHI,NDOT)

<table>
<thead>
<tr>
<th>Fortran Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM (floating)</td>
<td>The array to be contoured.</td>
</tr>
<tr>
<td>MX (integer)</td>
<td>X dimension of AM.</td>
</tr>
<tr>
<td>NY (integer)</td>
<td>Y dimension of AM.</td>
</tr>
<tr>
<td>FLO (floating)</td>
<td>Value of lowest contour line.</td>
</tr>
<tr>
<td>HI (floating)</td>
<td>Value of highest contour line.</td>
</tr>
<tr>
<td>FINC (floating)</td>
<td>Increment between successive contour line values:</td>
</tr>
<tr>
<td></td>
<td>1) If FLO and HI are both zero, they will be set to the minimum and maximum values of AM respectively.</td>
</tr>
<tr>
<td></td>
<td>2) If FINC is 0, it will be set to (HI-FLO)/16.</td>
</tr>
<tr>
<td></td>
<td>3) If FINC is negative, it will be set to (HI-FLO)/(-FINC).</td>
</tr>
<tr>
<td>NSET (integer)</td>
<td>1) If 0, SET and PERIM will be called by CALCNT.</td>
</tr>
<tr>
<td></td>
<td>2) If positive, SET and PERIM will not be called by CALCNT.</td>
</tr>
<tr>
<td></td>
<td>3) If negative, SET and PERIM will not be called by CALCNT and the legend &quot;contour from ___ to ___, contour interval ___&quot; will be written at YCRT address +22 rather than +34 as is done in the 0 and + cases.</td>
</tr>
</tbody>
</table>
CALCNT (cont'd)

Fortran Function (cont'd)

NHI (integer) 1) If 0, highs and lows will be marked with an "H" or "L" as appropriate, and the value of the high and low will be written under the symbol.

2) If positive, 3 digits and sign of each AM value will be written with the sign indicating the data point location.

3) If negative, neither of the above will be done.

NDOT (integer) A ten-bit octal constant designating the DASHLN pattern desired for contour lines on the map. (See DASHLN writeup, page 418.) If NDOT is set to 0, 1, or 1777, solid lines will be drawn.

Ascent Calling Sequence

RJ CALCNT
JP *+10
C0N AM
C0N MX
C0N NY
C0N FL0
C0N HI
C0N FINC
C0N NSET
C0N NHI
C0N ND0T

Error Message None

Storage 3410_8 including a 1520_8 word labeled common block / CONT / and subroutines SCAN, LINEAR, HILO and PTVALU.

Timing About 2 seconds for an average smoothness field dimensioned 40 x 70.
CALCNT (cont'd)

**Method**  
Linear interpolation between cells whose AM value locations are specified by subscripts; i.e., AM(1,1) is at the lower left corner of map; AM(MX, NY) is at the upper right corner. Scaling, if done by CALCNT \[ NSET = 0 \] makes each cell a square.

### IDIOT

**Fortran Function**

<table>
<thead>
<tr>
<th>CALL IDIOT (FLDX, FLDY, N, LTYPE, IPAT, LABX, LBY, LTIT, LFRAME)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLDX (floating)</td>
</tr>
<tr>
<td>FLDY (floating)</td>
</tr>
<tr>
<td>N (integer)</td>
</tr>
<tr>
<td>LTYPE (integer)</td>
</tr>
</tbody>
</table>
|                | \[
| \begin{array}{ccc}
| LTYPE & X \text{ plotting} & Y \text{ plotting} \\
| 1 & \text{linear} & \text{linear} \\
| 2 & \text{linear} & \text{log} \\
| 3 & \text{log} & \text{linear} \\
| 4 & \text{log} & \text{log} \\
| \end{array}
| \] |
| IPAT (integer) | A 10-bit octal constant defining the line pattern of the curve to be drawn. If IPAT = 1777B, the line will be solid. A typical dashed line is 1740B. For further discussion, see page 418. |
| LABX (integer) | A 4-word array of Hollerith information for labelling the abscissa. |
### Library Routines - Plotting

**Other Useful Routines**

**IDIOT (cont'd)**

<table>
<thead>
<tr>
<th>Fortran Function (cont'd)</th>
<th>LABY (integer)</th>
<th>LTIT (integer)</th>
<th>LFRAME (integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A 4-word array of Hollerith information for labelling the ordinate.</td>
<td>A 4-word array of Hollerith information for a main title.</td>
<td>A flag for frame advance. If LFRAME is</td>
</tr>
<tr>
<td></td>
<td>Note: The easiest way to generate these three arrays is to read one array per card with a 4A10 format. If the title is centered, it will be centered in the plot field.</td>
<td></td>
<td>1) zero - no frame advance will be given.</td>
</tr>
<tr>
<td></td>
<td>2) negative - a frame advance will be called before plotting is started.</td>
<td>3) positive - a frame advance will be given after plotting is completed.</td>
<td></td>
</tr>
</tbody>
</table>

**Ascent Calling Sequence**

<table>
<thead>
<tr>
<th>RJ</th>
<th>IDIÔT</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP</td>
<td>*+10</td>
</tr>
<tr>
<td>CÔN</td>
<td>FLDX</td>
</tr>
<tr>
<td>CÔN</td>
<td>FLDY</td>
</tr>
<tr>
<td>CÔN</td>
<td>N</td>
</tr>
<tr>
<td>CÔN</td>
<td>LTYPE</td>
</tr>
<tr>
<td>CÔN</td>
<td>IPAT</td>
</tr>
<tr>
<td>CÔN</td>
<td>LABX</td>
</tr>
<tr>
<td>CÔN</td>
<td>LAMY</td>
</tr>
<tr>
<td>CÔN</td>
<td>LTIT</td>
</tr>
<tr>
<td>CÔN</td>
<td>LFRAME</td>
</tr>
</tbody>
</table>

**Error Message**

None

**Storage**

10008
IDIOT (cont'd)

Purpose
To make a single subroutine call produce a complete annotated X-Y plot. IDIOT searches your field, rounds the min and max field values appropriately, draws graph paper with a suitable number of lines and plots and labels a curve on the graph.

Additional curves may be made on the same graph by the following sequence:

CALL DASHLN(IPATRN) to make identifiable line
CALL CURVE(FLDX,FLDY,N) plot the curve

IDIOT uses the routines ROUND and LGRD (in the package).

SUPMAP

Detailed writeups of this routine may be obtained from the secretary to the Computing Facility.

Fortran Function
CALL SUPMAP(JPROJ,POLAT,POLONG,ROT,PL1,PL2,PL3,PL4,INDS,JGRID)

Entry Points
SUPMAP, SUPCON, MAPLOT

Program Size
There are three subroutines:

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPMAP</td>
<td>1455</td>
</tr>
<tr>
<td>MAPLOT</td>
<td>2750</td>
</tr>
<tr>
<td>SUPCON</td>
<td>461</td>
</tr>
</tbody>
</table>

Purpose
To draw maps in a variety of projections with an option of world coast lines and USA political boundaries.
A number of utility subroutines written in Fortran and not on the library tape have been collected and are available for your use. They represent many of the most commonly used algorithms in numerical analysis. If you should require a program which is not on the following list and which would be an asset to the library, the library group would be pleased to work with you on the development of such a program.

Each routine has been classified into one of fourteen groups as follows:

1. Algebraic zeros
2. Approximation and curve fitting
3. Elementary functions
4. Fast Fourier transforms
5. Graphic display
6. Input-output
7. Matrix analysis
8. Miscellaneous
9. Non-algebraic zeros
10. Ordinary differential equations
11. Partial differential equations
12. Quadrature
13. Random number generators
14. Statistics

A listing and brief description of the available routines follow.
### ALGEBRAIC ZEROS

#### Roots of a Polynomial

**CUBIC**
Algebraic solution of a third-order polynomial with real coefficients.

**PROOT**
Newton-Bairstow technique for a polynomial with real coefficients (NYU).

**TRAUB**
Traub's globally convergent method for a polynomial with complex coefficients.

### APPROXIMATION

#### AND CURVE FITTING

#### Least Squares

**LINLSQ**
Computes $a_i$'s such that the data is approximated in the least squares sense by a function of the form $y = a_1 f_1(x) + a_2 f_2(x) + \ldots + a_n f_n(x)$ where the $f_i(x)$ are specified by the user.

**LSTSQR**
Computes the coefficients of a least squares polynomial approximation to the data.

**NOLLS**
Nonlinear least squares package. NOLLS computes the parameters $a_i$ such that the data $y_i$ is approximated in the least squares sense by a function of the form $y = f(x, a_1, a_2, \ldots, a_n)$. (Note that the difference between NOLLS and LINLSQ is that the coefficients $a_i$ can occur nonlinearly in NOLLS.)

### Rational Approximation

**RATION**
Produces a rational function (i.e., a function of the form $\frac{P(x)}{Q(x)}$ where $P$ and $Q$ are polynomials) which approximates an explicitly given function in the sense that the maximum relative error is minimized.
<table>
<thead>
<tr>
<th>Library Routines</th>
<th>Special Purpose Subroutines</th>
</tr>
</thead>
</table>

**APPROXIMATION AND CURVE FITTING (cont'd)**

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSP</td>
<td>Contour spline. Determines two spline curves ( x(s) ) and ( y(s) ) which describe a contour parametrically.</td>
</tr>
<tr>
<td>SPLINE</td>
<td>Cubic spline approximation of a function.</td>
</tr>
</tbody>
</table>

**ELEMENTARY FUNCTIONS**

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EI</td>
<td>Exponential integral (with 15-digit accuracy).</td>
</tr>
<tr>
<td>ERF</td>
<td>Error function.</td>
</tr>
<tr>
<td>BESLJ</td>
<td>Bessel function of first kind, ( J_n(x) ).</td>
</tr>
<tr>
<td>BESLY</td>
<td>Bessel function of first kind, ( Y_n(x) ).</td>
</tr>
<tr>
<td>BESLK</td>
<td>Modified Bessel function of first kind, ( K_n(x) ).</td>
</tr>
<tr>
<td>BESLI</td>
<td>Modified Bessel function of first kind, ( I_n(x) ).</td>
</tr>
<tr>
<td>EXP</td>
<td>Fast exponential (6-digit accuracy).</td>
</tr>
</tbody>
</table>

**FAST FOURIER TRANSFORMS**

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFA</td>
<td>Forward transform, where the number of real data points is a power of two.</td>
</tr>
<tr>
<td>FFS</td>
<td>Reverse transform (to be used in conjunction with FFA).</td>
</tr>
</tbody>
</table>

**Multi-dimensional**

<table>
<thead>
<tr>
<th>SUBROUTINE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOUR2</td>
<td>Forward or reverse transform where the number of complex data points in each dimension is a power of two.</td>
</tr>
<tr>
<td>FOURT</td>
<td>Forward or reverse transform with no restriction on the number of complex data points in any dimension.</td>
</tr>
</tbody>
</table>
Library Routines  
Special Purpose Subroutines

GRAPHIC DISPLAY

Contouring
CALCNT Contour plots on dd80.

Plotting
IDIOT Curve plotting on dd80.

INPUT-OUTPUT

RECIIN Buffered handling of a large number of small records
RECOT on tape, drum, or disk.
TAPECY See section 600 for detailed descriptions.
RPTOUT
RPTIN

MATRIX ANALYSIS

Eigenvalues and Eigenvectors of a Matrix
CMPXQR Eigenvalues and eigenvectors of a complex matrix.
REALQR Eigenvalues and eigenvectors of a real matrix.
RSYMRQR Eigenvalues and eigenvectors of a real symmetric
matrix.

Matrix Inversion
INVMTX Inverse of a matrix (Gauss elimination with full
pivoting).

Orthogonalization
ORTHO Orthonormalizes a system of independent vectors.
Library Routines

Special Purpose Subroutines

MATRIX ANALYSIS
(cont'd)

Reduction to Hessenberg Form (i.e., \( a_{ij} = 0 \) for \( i > j+1 \))

HESREL Householder reduction of a real matrix to upper Hessenberg form.

HESREM Householder reduction of a real symmetric matrix to tridiagonal form.

Solution of a System of Equations

BNDSLV Solves the matrix equation \( Ax = B \) where \( A \) is a banded \( n \times n \) real matrix (\( A \) banded means that there exists an integer \( M \geq 0 \) such that \( a_{ij} = 0 \) whenever \( |i-j| > M \)).

EQSLV Solves the matrix equation \( Ax = B \) where \( A \) is a real \( n \times n \) matrix.

TRIDI Solves the matrix equation \( Ax = B \) where \( A \) is a real tridiagonal matrix.

TRIDIP Solves the matrix equation \( Ax = B \) where \( A \) is a real tridiagonal matrix except for the first and last equations which describe the periodic behavior of the unknowns (can be used to solve differential equations with periodic boundary conditions).

Matrix Package

DECOMP Decomposes a matrix into upper and lower triangular parts.

SOLVE Solves the matrix equation \( Ax = B \) using the above decomposition of \( A \).

IMPRUV Iterative improvement of the above solution.
### Library Routines

#### MISCELLANEOUS

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDATE</td>
<td>Given a date, NDATE determines the number of days from a given base date. Using a different entry point, NDATE determines the date from the number of days from a base date.</td>
</tr>
<tr>
<td>PERNUT</td>
<td>Determines all permutations of a set of integers.</td>
</tr>
<tr>
<td>RENO</td>
<td>Renumbers Fortran statement numbers.</td>
</tr>
</tbody>
</table>

#### NON-ALGEBRAIC ZEROS

**Roots of Nonlinear Equations**

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAGUER</td>
<td>Laguerre's method for the zeros of a complex function.</td>
</tr>
<tr>
<td>NLEQS</td>
<td>Kiener's method for the solution of a nonlinear equation.</td>
</tr>
<tr>
<td>RTSRH</td>
<td>Binary search for the real zeros of a function.</td>
</tr>
</tbody>
</table>

#### ORDINARY DIFFERENTIAL EQUATIONS

**Solution of a System of Ordinary Differential Equations**

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZNDA</td>
<td>Runge-Kutta with error control.</td>
</tr>
<tr>
<td>NDA</td>
<td>Hamming's method.</td>
</tr>
<tr>
<td>RNGKUT</td>
<td>Runge-Kutta with no error control.</td>
</tr>
</tbody>
</table>

**Eigenvalues of Ordinary Differential Equations**

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODEIG</td>
<td>Solves eigenvalue problem for a linear system of ordinary differential equations.</td>
</tr>
</tbody>
</table>
PARTIAL DIFFERENTIAL EQUATIONS

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADI</td>
<td>Solution of Poisson's equation on a rectangle (alternating direction-implicit technique).</td>
</tr>
</tbody>
</table>

QUADRATURE

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALGUER</td>
<td>Laguerre quadrature on the interval $(0, \infty)$.</td>
</tr>
<tr>
<td>GAUS</td>
<td>Gaussian quadrature.</td>
</tr>
<tr>
<td>ROMB</td>
<td>Romberg integration.</td>
</tr>
<tr>
<td>SIMPNE</td>
<td>Simpson quadrature for non-equally spaced abscissa.</td>
</tr>
<tr>
<td>SIMPSON</td>
<td>Simpson quadrature for either an even or an odd number of points.</td>
</tr>
</tbody>
</table>

Multi-dimensional

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASELI</td>
<td>Monte Carlo technique using quasi-random numbers.</td>
</tr>
</tbody>
</table>

RANDOM NUMBER GENERATORS

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANF</td>
<td>Random numbers with uniform distribution (NYU).</td>
</tr>
<tr>
<td>RNDE</td>
<td>Random numbers with normal distribution.</td>
</tr>
<tr>
<td>STATISTICS</td>
<td>Subroutine</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>ASA</td>
<td>ASA</td>
</tr>
<tr>
<td>BMD04M</td>
<td>Discriminant analysis for two groups.</td>
</tr>
<tr>
<td>BMD06M</td>
<td>Canonical analysis.</td>
</tr>
<tr>
<td>BMD02R</td>
<td>Stepwise regression.</td>
</tr>
<tr>
<td>BMD05R</td>
<td>Polynomial regression.</td>
</tr>
<tr>
<td>BMD02S</td>
<td>Contingency table analysis.</td>
</tr>
<tr>
<td>BMD02V</td>
<td>Analysis of variance for factorial design.</td>
</tr>
<tr>
<td>CORR</td>
<td>Determines correlation between two series.</td>
</tr>
<tr>
<td>FACT</td>
<td>Factor analysis.</td>
</tr>
<tr>
<td>HARMON</td>
<td>Goertzel's algorithm is used to determine the coefficients of a trigonometric expansion.</td>
</tr>
<tr>
<td>POWERX</td>
<td>Computes the power spectrum of two series and the coherence, phase and gain.</td>
</tr>
</tbody>
</table>
In this section are included routines which can be used in various data processing problems. For consulting on data processing problems, including where to find the data, how to decode problem formats, tape information, volume compaction, etc., see Roy Jenne or Dennis Joseph.

The following routines are discussed:

1. Tape Copy and Edit
2. RPTOUT and RPTIN -- to pack and unpack reports on tape.
3. Other Data Handling Routines

**TAPE COPY AND EDIT**

(Not on system tape)

This is a main program used to copy, combine, and edit tapes. The routine includes a main driver program, a tape copy subroutine, and two user subroutines which may be modified for specific copy jobs.

The main program contains a LTH parameter which must be set to at least the maximum record length. Dimension IBUF by two times LTH.

**User Subroutines**

**USER**

The USER subroutine enables the user to examine each record and determine whether or not to copy the record. But if the error option has been set to 1, this subroutine is not called when there is a parity error. The entry to USER is as follows:
TAPE COPY AND EDIT (cont'd)

SUBROUTINE USER(IBUF,IWORDS,NFILE,NREC,ISTATE,NN)

IBUF The address of the first word of the current record.
IWØRDS The length of the current record.
NFILE Number of the current file.
NREC Number of records read in the current file.
ISTATE Status of current read.
0 = good read.
2 = parity error.

NN Flag set in USER to determine whether or not the record is to be copied.
0 = do not copy.
1 = copy this record.

USERF

The USERF subroutine allows the user to take control after each file in a multiple file copy.

SUBROUTINE USERF(NRSUM,NRECS,NFILE,NF)

NRSUM The total number of records read under control of the current control card.
NRECS Number of records in this file.
NFILE Number of the file just completed.
NF A flag set for action taken after this call.
0 = continue copy with next file and do not write an end file on the output tape.
1 = same as 0, but write an end file on the output tape.
2 = go to next control card and do not write an end file on the output tape.
3 = same as 2 but write an end file on the output tape.
TAPE COPY AND EDIT (cont'd)

For simple copies of a given tape the USER routine would consist simply of the card NN=1. The USERF routine would consist simply of the card NF=1.

Control Cards

Format

Main program control is accomplished through the use of control cards in the data section of the program. The general card format is as follows:

<table>
<thead>
<tr>
<th>Card Column</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Control word (left-justified)</td>
</tr>
<tr>
<td>6-10</td>
<td>NREC number of records or files</td>
</tr>
<tr>
<td>21-22</td>
<td>UNIT1 a logical tape unit (1-10)</td>
</tr>
<tr>
<td>27-28</td>
<td>UNIT2 a logical tape unit (1-10)</td>
</tr>
<tr>
<td>38</td>
<td>Mode of tape on UNIT1</td>
</tr>
<tr>
<td>40</td>
<td>Mode of tape to be written on UNIT2</td>
</tr>
<tr>
<td>48</td>
<td>Type of tape on UNIT1</td>
</tr>
<tr>
<td>50</td>
<td>Type of tape to be written on UNIT2 (always zero)</td>
</tr>
<tr>
<td>57</td>
<td>Error option</td>
</tr>
<tr>
<td></td>
<td>1 = copy only good records and delete those with parity errors</td>
</tr>
<tr>
<td></td>
<td>2 = copy all records</td>
</tr>
</tbody>
</table>
### TAPE COPY AND EDIT (cont'd)

**Mode and Type Options**

<table>
<thead>
<tr>
<th>Mode and Type Options</th>
<th>Mode and type options are the same as those listed under IPRC:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>MODE</strong> 0 = even parity (BCD mode), no character conversion</td>
</tr>
<tr>
<td></td>
<td>1 = odd parity (binary mode), no character conversion</td>
</tr>
<tr>
<td></td>
<td>2 = even parity (BCD mode), conversion of external BCD to display code (or vice versa if writing a record).</td>
</tr>
<tr>
<td></td>
<td>3 = odd parity (binary mode), conversion as for 2</td>
</tr>
<tr>
<td></td>
<td><strong>NTYPE</strong> 0 or 1 = read (or write) a record in the NCAR system format.</td>
</tr>
<tr>
<td></td>
<td>2 = read a record from another computer. For physical records longer than 512 sixty-bit words, you will only get the first 512 words.</td>
</tr>
<tr>
<td></td>
<td>3 = read a record of any length that was written by another computer. (This option does not allow character conversion.)</td>
</tr>
</tbody>
</table>

**Note:** Add 4 to any of the above options to ignore read parity error. The system will not try to correct this error for you. Thus a type 7 is the same as a type 3, but with no reread on parity errors.

### Control Words

<table>
<thead>
<tr>
<th>Control Words</th>
<th>The valid control words and their functions are listed below.</th>
</tr>
</thead>
<tbody>
<tr>
<td>REM</td>
<td>Remark card which can contain any remarks in columns 6-80.</td>
</tr>
<tr>
<td>Rew</td>
<td>Rewind UNIT1.</td>
</tr>
<tr>
<td>EOF</td>
<td>End file UNIT1.</td>
</tr>
<tr>
<td>COPY</td>
<td>Copy NREC records (or one file, whichever comes first) from UNIT1 to UNIT2 with the modes, types, and error option as specified.</td>
</tr>
</tbody>
</table>
TAPE COPY AND EDIT (cont'd)

CFILE  Copy NREC files as for COPY. The copy will continue until NREC files have been copied or the copy is terminated in the USERF routine.

SKIP   Skip NREC records (or one file, whichever comes first) on UNITI. The output parameters must be specified even though they are not used.

DONE   The job is terminated.

Each control card is processed sequentially and is printed before it is processed.

Copy Output

The output from the copy will indicate if there are any parity errors, the number of records copied, and the range of lengths in the records copied.

Listing

Following is a sample output from a tape copy which for reasons of illustration is unusually complicated and contains a high number of errors.

```
REM COPY OUTPUT EXAMPLE
REM CARD COLUMNS
REM 10 22 28 40 50 57
REM 01
REM 0
CFILE 2 FLS FROM 1 TO 8 MODE 1,1 TYPE 0,0 ERR 2
RECORD 57 IS REDUNDANT.
TROUBLE ON A WRITE 57 57 0 0 2
RECORD 16 IS REDUNDANT.
158 RECORDS READ WITH LENGTHS OF 0 TO 400 WORDS.
TROUBLE ON A WRITE 16 16 0 0 2
97 RECORDS READ WITH LENGTHS OF 0 TO 400 WORDS.
COPY 59 REC FROM 1 TO 8 MODE 1,1 TYPE 0,0 ERR 1
RECORD 31 IS REDUNDANT.
94 RECORDS READ WITH LENGTHS OF 0 TO 400 WORDS.
SKIP 25 REC FROM 1 TO 8 MODE 1,1 TYPE 0,0 ERR 2
29 RECORDS READ WITH LENGTHS OF 256 TO 258 WORDS.
COPY 9999 REC FROM 1 TO 8 MODE 1,1 TYPE 0,0 ERR 2
57 RECORDS READ WITH LENGTHS OF 256 TO 258 WORDS.
WEOF 8
DONE
```
TAPE COPY AND EDIT (cont'd)

Explanation

This set of cards with the simple user routines described previously would copy two files from the tape assigned to logical unit 1 to the tape assigned to logical unit 8. Then it would copy 50 records, skip 25 records, and copy to the next end file. The print lines beginning with control words are images of the control cards while the other print lines indicate the results of the previous control card. Note that the first file had one redundancy (record with a parity error) at record number 57 of that file. The second file had a parity at record number 6. The first file contained 138 records and the second 97 records. The copy 50 record had the error option set to 1 and encountered a parity at record number 31. Therefore this record was not copied. The copy 9999 record card was terminated when an end file was encountered after 37 records all 250 words long. The trouble on a write message here indicates an attempt to write a zero length record because of the previous parity error on the read. The print indicates number of record in, number of record out, length in, length out, and status of the write.
RPTOUT and RPTIN
(not on system tape)

Purpose
RPTOUT is used to pack short variable length logical records into larger physical records on tape. The records are under checksum control. Logical records can be 1 to 352 words long, but the first 12 bits of each logical record must be reserved for use by RPTOUT. RPTIN is used to read logical records from a tape made by RPTOUT. Several tapes can be written or read at once, but each must have a separate buffer.

Method
In general, the I/O operators are used in your program ignoring that the records are blocked. The exceptions to this are as follows:

1) A special buffer dimensioned 360 must be set aside by your program for use by RPTIN/RPTOUT in blocking and unblocking the logical records (reports). There must be a separate buffer for each unit in use at the same time.

2) There is no backspace command.

3) Before putting an end file on a tape, or otherwise terminating writes on a tape, a special RPTOUT command with JL=2 is required (see below).

4) After a rewind or before using an NFUB buffer for another tape, NBUF(1) must be set to zero, and the counters in NBUF may be reinitialized if desired.
RPTOUT and RPTIN (cont'd)

**Write Logical Record**

To write out a logical record,

CALL RPTOUT (NUNIT, NBUF, LOCRPT, NWDS, JL)

- **NUNIT** Tape numbers 1-10 (not 5 or 6)
- **NBUF** An array in which the records are built:
  - Dimension NBUF(360)
  - NBUF(1) Must be 0 before the first write on a tape. RPTOUT will start a record when it sees the 0.
  - NBUF(2-4) Must be cleared if you need to reinitialize the counters.
  - NBUF(2) Will have a count of logical records output.
  - NBUF(3) Will have a count of physical records output.
  - NBUF(4) Will be a count of the words output. But words in reports = NBUF(4) - 2*NBUF(3).
- **LOCRPT** Location of the report for output. The first 12 bits of the report will be used by RPTOUT and RPTIN. Any data contained in the first 12 bits of the first word will be destroyed.
- **NWDS** Number of words in the report.
- **JL**
  - = 0, output this report.
  - = 2, no report to output; output the reports in NBUF onto the tape. This is used to output the last buffer onto the tape.

**Error Messages**

Bad physical write, unit, physical record, length, status. This usually indicates some problems with the physical tape, tape drive, or machine.
RPTOUT and RPTIN (cont'd)

Error Messages (cont'd)  Bad logical length, unit, logical record #, length. The user tried to write either a zero length record or a record longer than 352 words.

Read Logical Record  To read a logical record,

CALL RPTIN (KUNIT,KBUF,KLOC,KWDS, JJ,KIMAX,JEOF)

KUNIT  Tape 1-10 (not 5 or 6)

KBUF  An array to use for unpacking the records. Dimension KBUF(360) KBUF(1) must be set to 0 before the first read. RPTIN will start a new read when it sees the 0.

KBUF(2-6) Must be cleared when desired.

KBUF(2) Will have a count of logical records read.

KBUF(3) Will have a count of physical records read.

KBUF(4) Will have a count of words read from the tape.

KBUF(5) Will have the words in this record.

KBUF(6) Will be the same as JEOF.

KLOC  Array to put report in.

KWDS  The total number of words in this report.

JJ  = 1, return only two words of this report and do not move on to the next report (except for EOF case).

= 10, return full report up to a maximum of KIMAX words and position on next report. This is the option normally used.

KIMAX  Maximum number of words in a report to move the KLOC.
RPTOUT and RPTIN (cont'd)

Read Logical Record (cont')

\[ \text{JE0F} \]

\[ = 0, \text{ good report returned.} \]

\[ = 1, \text{ EOF} \]

\[ = 2, \text{ report returned from a record with a bad checksum} \]

Error Messages

Short physical record, unit, physical record, length, expected length, status. Bad tape read or records were not created by RPTOUT. This diagnostic often simply indicates noise in a record gap.

Bad physical length, unit, physical record, length, expected length, status. Bad tape read or records were not created by RPTOUT.

Bad checksum, unit, physical record, length, expected length, status. Bad tape read or records were not created by RPTOUT.

Bad logical length, unit, logical record, length. Probably the result of one of the previous errors.
OTHER DATA HANDLING ROUTINES

If no further manual writeup is available, see Roy Jenne or Dennis Joseph for further details on these programs or for information regarding other similar programs which might exist.

RDTAPE and WRTAPE

Tape input/output routines similar to Fortran BUFFERIN/BUFFEROUT, which provide some facilities not available under the Fortran operations (see IOPROC under Library Routines - Utility).

SORTUU and SORTBB

Tape sort routines which operate on unblocked or RPTOUT format tapes and sequence the records in a file. The part of the record used to sequence the records is completely flexible and is determined in a user-written subroutine.

GBYTES and SBYTES

Routines for manipulating bits in single bytes or arrays of equal length bytes. (See GBYTES under Library Routines - Utility.)

GBITS and SBITS

Routines for manipulating bits under something similar to format control.

UBLOK and UZBLOK

These routines facilitate unblocking of fixed length logical records from tapes with such a blocking scheme.
OTHER DATA HANDLING Routines (cont')

CDTAPE and CDTAPEB

These routines put punched cards onto tape in RPTOUT format (see RPTIN/RPTOUT). The input cards may be Hollerith, coded or column binary.

CHCONV

A character conversion routine for converting from one 6-bit code to another 6-bit code such as from BCD to display code.

CDS360

A routine to convert from IBM 360 punched card codes to NCAR-compatible punched card codes.

REPBIN

This and other routines are available for reproducing card decks regardless of format. The routine is available in the machine room.

CORFOR

(Corrects Fortran written tapes)  A program to copy records created with the Fortran WRITE(NUNIT) statement. The Fortran READ(NUNIT) routine terminates execution on a tape error, and the CORFOR program facilitates recovery or deletion of these bad records.

DREC

(Dump records) A tape-to-printer routine for tape dumps.
**CENTRAL MONITOR**

**STORAGE**
The Central Monitor (CM) occupies the first 30008 cells of all executing programs.

**ERROR MESSAGES**
- BCD RECORD ENCOUNTERED DURING BINARY READ
- BINARY RECORD ENCOUNTERED DURING BCD READ
- ASSIGN LOGICAL TAPE

**PURPOSE**
The CM is called to pass all I/O requests from an executing central memory program to the peripheral monitor in PPO. Various other functions are also performed by CM; however, only the use of CM as an I/O supervisor is described here.

**REGISTERS DESTROYED**
CM will destroy the X7, A1, A2, A3, A4, A5 and A6 registers.

**CALLING SEQUENCE**
All I/O calls from a central memory program are made by calling CM with a calling sequence in the following form:
CALLING SEQUENCE (cont'd)

<table>
<thead>
<tr>
<th>RJ</th>
<th>CM</th>
<th>(CM is always location 20418)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP</td>
<td>RETURN</td>
<td>(return location)</td>
</tr>
<tr>
<td>EQ</td>
<td>ØP</td>
<td>(operation code)</td>
</tr>
<tr>
<td>EQ</td>
<td>CV</td>
<td>(conversion code)</td>
</tr>
<tr>
<td>EQ</td>
<td>FWA</td>
<td>(first word address)</td>
</tr>
<tr>
<td>EQ</td>
<td>IWA+1</td>
<td>(last word address + 1)</td>
</tr>
<tr>
<td>EQ</td>
<td>UNIT</td>
<td>(unit address)</td>
</tr>
<tr>
<td>EQ</td>
<td>STATUS, BN</td>
<td>(status address and tape format)</td>
</tr>
</tbody>
</table>

RETURN

RETURN is the address to which control passes following the call to CM. This may simply be +4 to jump around the calling sequence and continue the program.

OP

OP is the operation code. The peripheral processors interpret them as:

- SFF    EQU 1       Forward space one file
- SFB    EQU 2       Backspace one file
- WFM    EQU 3       Write an end of file
- RWL    EQU 4       Rewind
- RWU    EQU 5       Rewind and unload
- FSP    EQU 6       Forward space one record
- BSP    EQU 7       Backspace one record
- RFC    EQU 10      Read BCD (even parity)
- RFB    EQU 11      Read binary (odd parity)
- WRC    EQU 12      Write BCD
- WRB    EQU 13      Write binary

If \(4000_8\) is added to any of the above operation codes, the CM routine will retain control until the operation is completed. In fact, CM will signal the peripheral monitor to do an exchange jump to another central program. An operation code of \(4000_8\) with a call that includes the status address of a previous call will also result in an exchange jump to another central program unless the status word indicates the operation is already completed. Thus the \(4000_8\) modifier serves as a WAIT operation.
CALLING SEQUENCE

OP (cont'd)

code. If an operation code occurs with no wait, CM will pass the request to the peripheral monitor and then return control to the calling program before the operation has been completed.

CV

CV is the conversion code and is needed only for BCD tape operations. The central processor uses "display code" and if a tape is read or written in the BCD mode, a conversion code must be specified. If $CV = 1$, it will cause a conversion when reading from the BCD on the tape to DPC in core. If $CV = 2$, it will cause a conversion when writing from DPC in core to BCD on the tape. Set $CV = 0$ for binary tapes.

FWA

FWA is the first word address of the data to be read or written.

LWA

LWA is the address of the last word of data to be read or written. (The argument is LWA + 1.)

UNIT

UNIT is the address of a core cell containing the logical unit number referenced by the call. Logical units 1 - 31 are assigned with an *ASSIGN control card. Unless an *ASSIGN control card has been used to assign logical unit 5 or 6, these units will be assigned as the standard input unit (card reader) and the standard output unit (printer) respectively. This is done to allow their use in the Fortran statements:
CALLING SEQUENCE
(continuation)

UNIT (continuation)

READ INPUT TAPE 5, format, list
WRITE OUTPUT TAPE 6, format, list

without the presence of *ASSIGN cards. Four other units which need not be assigned are

718 printer
728 card reader
738 dd80
748 card punch

STATUS

STATUS is the address of a cell in which the peripheral processor actually performing the I/O operation can return status bits for the operation to the central program. The lower 18 bits contain the number of central processor words transmitted. The upper 24 bits are set for the following conditions:

<table>
<thead>
<tr>
<th>Status Replies</th>
<th>Bits 59-48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation unsuccessful (bit 59)</td>
<td>4XXX</td>
</tr>
<tr>
<td>Load point on tape (bit 55)</td>
<td>X2XX</td>
</tr>
<tr>
<td>End of tape (bit 54)</td>
<td>X1XX</td>
</tr>
<tr>
<td>Unrecoverable read error (bit 52)</td>
<td>XX2X</td>
</tr>
<tr>
<td>Short read (bit 51)</td>
<td>XX1X</td>
</tr>
<tr>
<td>End of file (bit 49)</td>
<td>XXX2</td>
</tr>
<tr>
<td>Calling error (FWA &gt; LWA, FWA &gt; FL, LWA &gt; FL) (bit 48)</td>
<td>XXX1</td>
</tr>
</tbody>
</table>
### CALLING SEQUENCE

#### STATUS (cont'd)

<table>
<thead>
<tr>
<th>Status Replies for Tape Operation</th>
<th>Bits 47-36</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved (by other channel) (bit 47)</td>
<td>4XXX</td>
</tr>
<tr>
<td>Vertical or longitudinal parity error (bit 46)</td>
<td>2XXX</td>
</tr>
<tr>
<td>End of operation (bit 45)</td>
<td>1XXX</td>
</tr>
<tr>
<td>Lost data (bit 44)</td>
<td>X4XX</td>
</tr>
<tr>
<td>Density (&quot;1&quot; in bit 43 indicates 800 BPI) (bit 43)</td>
<td>X2XX</td>
</tr>
<tr>
<td>Density (&quot;1&quot; in bit 42 indicates 556 BPI; &quot;0&quot; in bits 6 and 43 indicates 200 BPI) (bit 42)</td>
<td>X1XX</td>
</tr>
<tr>
<td>End of tape (bit 41)</td>
<td>XX4X</td>
</tr>
<tr>
<td>Load point (bit 40)</td>
<td>XX2X</td>
</tr>
<tr>
<td>File mark (bit 39)</td>
<td>XX1X</td>
</tr>
<tr>
<td>Write enable (bit 38)</td>
<td>XXX4</td>
</tr>
<tr>
<td>Channel and/or Read/Write control and/or Unit Busy (bit 37)</td>
<td>XXX2</td>
</tr>
<tr>
<td>Ready (bit 36)</td>
<td>XXX1</td>
</tr>
</tbody>
</table>

If STATUS = 0, no status will be returned by the peripheral processor. If a STATUS cell is specified, it is cleared before CM passes the request to the peripheral monitor.

BN is a flag that specifies tape format options if the unit is assigned to a tape. If BN = 0, the tape is in the standard system format with checksums. If BN = 1, the tape will not be checksummed. If BN = 2, the tape is composed of non-system records. The tape formats are described in detail in the Fortran manual.
Central Monitor

CM handles the blocking and unblocking of all information for the standard input file and output file on the disk. The input file may be read in either the BCD or binary mode. The output file may be written in either mode. The only legal OP codes on these units are RFC or RFB for the input file and WRC or WRB for the output file. No backspace, rewind or other non-transmitting operation is permitted.

Calls referencing units 5 and 72 are reading the input unit, and a unit record or card image is unpacked from the input buffer and placed in the area starting at FWA. Only the number of words requested from the card are transmitted to the program. The buffer pointers are then moved to the start of the next card.

Data from calls referencing units 6, 71, 73, and 74 are flagged with the unit name and placed in the output buffer. The buffer with all data types is written on logical unit 71. If the dd80 has been assigned for use during execution phase, data from calls referencing unit 73 is sent directly to the dd80. If cards are to be punched, they must be on unit 74 and the user must be careful to transmit exactly 16 words per card. If the call is not a multiple of 16 words, the cards will be incorrectly punched.

CM handles the input and output files in a wait mode. The data will always be moved before control is returned to the main program. If an end of file was sensed on the input medium, the status cell will be set with an EOF flag. Otherwise, CM does not set anything in the status cell for calls referencing these units. The conversion code CV and the BN options are ignored.
The following points should be noted:

1. The calling sequence to CM may not be modified until after an I/O operation is complete, since the peripheral processor performing the operation uses the calling sequence in central memory.

2. The following call will flush the output buffer and destroy all registers:

   SB1
   RJ
   1
   CM+2

3. The following call will reset the input buffer pointers so that the next read will require a disk access. This will also destroy all registers:

   SB1
   RJ
   1
   CM+4
The following code could be used to read one card image from a BCD tape written on another computer and mounted on logical unit 3:

```
CM EQU 2041B
RFC EQU 10B
:
RJ CM Return jump to CM
JP *+4 Jump around calling sequence
EQ RFC+4000B Read BCD and wait
EQ 1 Convert from BCD to DPC
EQ CARD FWA
EQ CARD+8 IWA+1
EQ UNIT Address of cell containing logical unit number
EQ STAT,B2 Tape contains non-system records
:
UNIT CON 3 Logical unit 3 must be assigned
STAT CON 0 Storage for status word
CARD BSS 8 Storage for card image
```
The following subroutine could be used to write a comment on the printer:

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>COMNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CM</td>
<td>EQU</td>
</tr>
<tr>
<td>WRC</td>
<td>EQU</td>
</tr>
<tr>
<td>COMNT</td>
<td>CON</td>
</tr>
<tr>
<td></td>
<td>RJ</td>
</tr>
<tr>
<td></td>
<td>JP</td>
</tr>
<tr>
<td></td>
<td>EQ</td>
</tr>
<tr>
<td></td>
<td>EQ</td>
</tr>
<tr>
<td></td>
<td>EQ</td>
</tr>
<tr>
<td></td>
<td>EQ</td>
</tr>
<tr>
<td></td>
<td>EQ</td>
</tr>
<tr>
<td>OUT</td>
<td>CON</td>
</tr>
<tr>
<td>STAT</td>
<td>CON</td>
</tr>
<tr>
<td>MSG1</td>
<td>DPC</td>
</tr>
<tr>
<td>MSG2</td>
<td>EQU</td>
</tr>
<tr>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
EXECUTION DIAGNOSTICS

A list of diagnostics is printed by the monitor if a program terminates during execution. The routine in which the fault occurred is printed first. A brief error message is printed, followed by the octal location in the program from which the subroutine was entered.

Example:

IBAIEX

RESULT EXCEEDS 2 TO 48 FROM LOC 003021

The routine which had the fault during execution of the job was IBAIEX (in Fortran, I**J, the mnemonics of the name suggest integer base to an integer exponent). The message says that the answer is larger than $2^{48}$. The location in core from which this routine was called is 3021. If you subtract 3000, the routine origin in the loader map, from 3021, the result, 21, says that the jump to IBAIEX is 21 locations into the program which started at 3000. The location field in a Fortran program printout is the left-most column. It is a "relative address" since these locations start at 0 for each routine and are relative to the routine origin found in the loader map.
GENERAL DISCUSSION (cont'd)

An alphabetical listing of the subroutines, with their associated messages, follows with some comments to clarify the diagnostics under the section on utility routines. All of these messages are printed as a result of a jump to SYSERR. A typical exit to SYSERR is in IBAIEX. IBAIEX contains the following code on an error exit. If bits 9-26 of SYSERR are non-zero, the value of that field will be printed as an actual number following the last word of the error message.

SX1  98
SA2  IBAIEX
AX2  30
SX2  X2-1
LX2  9
BX6  X1*X2
SA2  SYSERR
MXO  30
BX2  XO*X2
BX6  X6+X2
SA6  SYSERR
EQ   SYSERR
# Mathematical Library Functions

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALOG</td>
<td>ARG ZERO OR NEG</td>
</tr>
<tr>
<td>ALOG10</td>
<td>ARG ZERO OR NEG</td>
</tr>
<tr>
<td>ASIN/ACOS</td>
<td></td>
</tr>
<tr>
<td>ATAN</td>
<td>INDEFINITE OR INFINITE ARG FROM LOC</td>
</tr>
<tr>
<td>ATAN2</td>
<td>INDEFINITE OR INFINITE ARG OR X=Y=0</td>
</tr>
<tr>
<td>CBAIEX</td>
<td>ZERO TO ZERO OR NEG POWER</td>
</tr>
<tr>
<td>CLOG</td>
<td>ARG (0.,0.) FROM LOC</td>
</tr>
<tr>
<td>DATAN</td>
<td>INDEFINITE OR INFINITE ARG FROM LOC</td>
</tr>
<tr>
<td>DATAN2</td>
<td>X=Y=0 OR INDEFINITE OR INFINITE ARG FROM LOC</td>
</tr>
<tr>
<td>DBADEX</td>
<td>NEG BASE WITH REAL EXP FROM LOC</td>
</tr>
<tr>
<td>DBAREX</td>
<td>NEG BASE WITH REAL EXP FROM LOC</td>
</tr>
<tr>
<td>DCOS</td>
<td>ABS(ARG) = PI X 2 to 94 FROM LOC</td>
</tr>
<tr>
<td>DEXP</td>
<td>ARG GREATER THAN 743. FROM LOC</td>
</tr>
<tr>
<td>DLOG</td>
<td>ARG ZERO OR NEG FROM LOC</td>
</tr>
<tr>
<td>DLOG10</td>
<td>ARG ZERO OR NEG FROM LOC</td>
</tr>
<tr>
<td>DMOD</td>
<td>INDEF OR INF ARG, X2 = 0, OR INTEG PART TOO LARGE</td>
</tr>
<tr>
<td>DSIN</td>
<td>ABS(ARG) = PI X 2 to 94 FROM LOC</td>
</tr>
<tr>
<td>DSQRT</td>
<td>NEGATIVE ARGUMENT FROM LOC</td>
</tr>
<tr>
<td>EXP</td>
<td>ARG IN EXPF .GT. 741. FROM LOC</td>
</tr>
<tr>
<td>IBAIEX</td>
<td>RESULT EXCEEDS 2 TO 48 FROM LOC</td>
</tr>
<tr>
<td>SIN/COS</td>
<td>ARGUMENT TOO LARGE, LOC</td>
</tr>
<tr>
<td>SQRT</td>
<td>NEGATIVE ARGUMENT IN SQRT FROM LOC</td>
</tr>
<tr>
<td>RBADEX</td>
<td>INDEF OR INF ARG FROM LOC</td>
</tr>
<tr>
<td>RBAIEX</td>
<td>ZERO BASE TO ZERO OR NEG EXPONENT FROM LOC</td>
</tr>
<tr>
<td>RBAREX</td>
<td>EXPONENT OVERFLOW OR NEG BASE IN X TO Y FROM LOC</td>
</tr>
</tbody>
</table>
### Utility Routines

<table>
<thead>
<tr>
<th>Routine Name</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ACGOER</strong></td>
<td>ERROR IN COMPUTED GO TO</td>
</tr>
<tr>
<td></td>
<td>The program has specified a number which is less than the first branch or greater than the Nth branch and a BRANCH TO statement has gone outside the specified range.</td>
</tr>
<tr>
<td><strong>ASCENT</strong></td>
<td>BREAKDOWN BUFFERS EXCEEDED</td>
</tr>
<tr>
<td></td>
<td>More than 20 constants appear in the variable field in a statement of the following type: ( A \text{ CON } X,A,B,C,D )</td>
</tr>
<tr>
<td></td>
<td>CHECKSUM ERROR ON COSY CARD</td>
</tr>
<tr>
<td></td>
<td>The original checksum on the Cosy card is different from the checksum recalculated at the time of a read.</td>
</tr>
<tr>
<td></td>
<td>COMMON BLOCK TABLE OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>The program has specified more than 20 common block names.</td>
</tr>
<tr>
<td></td>
<td>ENTRY POINT TABLE OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>More than 40 entry points have been specified.</td>
</tr>
<tr>
<td></td>
<td>EXTERNAL TABLE OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>An Ascent program may specify only 20 subroutine names in external statements.</td>
</tr>
<tr>
<td></td>
<td>ILLEGAL INSERT DELETE REPLACE CARD</td>
</tr>
<tr>
<td></td>
<td>The cosy card number(s) must be positive, non-zero integer(s) and the first number must be smaller than the second number.</td>
</tr>
<tr>
<td></td>
<td>INSERT TABLE OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>125 cards may be inserted without increasing the field length using a *LIMIT, C=60000 card.</td>
</tr>
<tr>
<td></td>
<td>MACRO DICTIONARY OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>The limit on number of macros is 50. The program has exceeded this limit.</td>
</tr>
<tr>
<td></td>
<td>MACRO PARAMETER TABLE OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>Too many parameters are used in a macro definition.</td>
</tr>
<tr>
<td></td>
<td>MACRO SKELETON TABLE OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>The macro skeletons have exceeded 30 words. A skeleton table is a framework for a macro. When the macro is inserted in the Ascent program, variables and parameters are substituted in the skeleton structure. It is hard to estimate the number of words in a macro skeleton as more than one instruction per word may be packed.</td>
</tr>
</tbody>
</table>
NO ENTRY POINT

Every deck must have at least one card with the pseudo op entry. No starting location is specified for entry into an Ascent program on an ENTRY card.

POINTER TABLE OVERFLOW

There may be no more than 200 INSERT, DELETE and REPLACE cards.

SYMBOL TABLE OVERFLOW

The number of labels or symbols that appear in columns 1-7 exceed 1100.

BACKSP

POSITIONING ERROR

Two different tape formats are written; the standard format from OUTPTC or OUTPTB, and the special format written by BUFFER OUT. The format is assumed to be standard unless a BUFFER statement has been executed prior to the backspace. Attempts to backspace a tape in the wrong format will cause a positioning error diagnostic.

BUFFEI/O (Buffer In and Buffer Out)

TOO MANY UNITS

The number of different units in one program is limited to seven for BUFFER statements.

UNASSIGNED UNIT

The logical unit specified in a BUFFER statement has not been requested on an *ASSIGN card. Tapes 5 and 6 may not be used.

CM

ASSIGN LOGICAL TAPE (i is the unit number)

An assign card is missing from the deck or incorrectly punched. Check the logical unit number on the assign card and the tape unit specified in the Fortran program. If the unit number (i) is missing from the comment printed on the output, an attempt was made to reference logical unit zero, and was not assigned in a *ASSIGN card.

BINARY RECORD ENCOUNTERED DURING BCD READ

A binary card occurs without a *BINARY control card, or a Cosy deck is not preceded by a COSY card. This occurs only when reading the input file.

BCD RECORD ENCOUNTERED DURING BINARY READ

An Ascent card may have occurred in a Cosy deck, or a Fortran card may be in a binary deck. This occurs only when reading an input file.
DD80 FRAME LIMIT EXCEEDED
The frame count exceeds frames specified on the *LIMIT card. If the *LIMIT card is not in the deck, the frame count is assumed to be 100.

PAGE LIMIT EXCEEDED (or PAGE LIMIT FOR PRINTER 2 EXCEEDED)
The page count exceeds pages specified on the *LIMIT card. If the *LIMIT card is not in the deck then the page count is assumed to be 10. If 6 or 57 have been assigned to the dd80, a frame (not page) count will be monitored.

PUNCH CARD LIMIT EXCEEDED
Either the *LIMIT card specification for PU is too small or the *LIMIT card is not in the deck and the program exceeded 100 cards (the assumed limit).

DRUM

DRUM REJECT ERROR
This is a hardware error. Resubmit your job.

DRUM/DISK

TABLE OVERFLOW
Software tables are too full. Resubmit your job.

PARITY ERROR
This is a hardware error. Resubmit your job.

STATUS LOCATION ERROR
The location at which the status is to be stored is out of bounds.

DUMP

PERIPHERAL DUMP CALLED FROM
In order to get a dump, in addition to the panel, a *TRAP monitor control card must be in the deck.

FLASH

FLASH BUFFER OVERFILL
A FLASH1 call has initiated plotter instructions in one of your arrays. More words of plotter instructions have been generated than array space provides.

FLASH1

TRIED TO START FLASH WHILE IN FLASH
FLASH1 may not be called twice without an intervening FLASH2 call.
INPUTB
Input B is used by a READ i or READ TAPE i instruction.

TAPE ERROR IN INPUTB OR OUTPTB, CALLED FROM
It is possible that the program has read past the end of the data on a tape or is reading a BCD tape in binary mode. The assign card may have the wrong density or you may have the wrong tape label on the assign card. It is also possible that the tape may be badly worn or torn.

INPUTC/S
This routine is called whenever a READn,L READ(i,n)L or READ INPUT TAPE i,n,L is executed. INPUTS is called whenever DECODE(c,m,V)L is executed. The format statement is not scanned during compilation. It is scanned at execution time.

DATA OVERFLOW - EXPONENT OUT OF RANGE
Check for the characters other than D, E, +, - in an E field to see if the exponent is too large. It may not be right-adjusted.

END FILE ON INPUT (no panel is printed)
This diagnostic occurs if the program tries to execute a read statement when there is no more data on the input file. Check the data for card count on large arrays or indexing errors in the read statement. Many programmers terminate on an END OF FILE by branching back to a READ instead of counting data sets. In this case there is no programming error; the message is informative.

EXCEEDED RECORD SIZE
A READ statement may specify 80 card columns or 50 characters from BCD tape for input. Examine FORMAT for too many repeated groups or omitting a slash.

EXPONENT TOO LARGE ON DATA INPUT, LOCATION
A data card has an exponent which is too large.

HOLLERITH FORMAT WITH LIST
No field specification for the list is contained in the format statement.

ILLEGAL DATA ENCOUNTERED FROM LOC
Look for a decimal point in an I field, a decimal digit in an O field, or an alphabetic or illegal special character in an I,O,E,D or F field. The indexing on the read statement may be too large or the data card count is wrong, and the card read does not go with this FORMAT statement.

ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT
A number specification has been made using a character other than X,I,A,R,F,E,D,O,H,*,L,P,+. Look for keypunch errors. A wild index or too small dimension statements may have caused some of core to be overwritten. Also check the count in an H specification.
INTEGER TOO LARGE ON DATA INPUT, LOCATION
   An integer greater than $\pm 2^{33} - 1$ is specified on input.

MULTIPLE DECIMAL POINTS IN DATA
   An illegal data field has two or more decimal points.

PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
   Examine parentheses in FORMAT statements. You have more left than right parentheses.

TAPE ERROR DURING READ OPERATION
   The program attempted to read a binary tape in BCD mode or there was a possible parity error.

ZERO FIELD WIDTH IN FORMAT STATEMENT
   There is a zero field width (like FO.3, or IO). Look for commas omitted after the specification or an error in the Hollerith count. Also look for indexing errors or dimension statement errors which may have "clobbered" the FORMAT statement.

INPUT MONITOR
   There are a number of errors which are detected immediately as the cards are read in by the input monitor. No subroutine or location is noted, as the job is terminated on input; one of the following messages will print out:

   BCD CARD IN THE BINARY DECK
   FORTRAN AFTER ASCENT
   JOB OR SEQUENCE CARD MISSING
   SEQUENCE CARD IN WRONG PLACE
   TAPE 3 NEEDED FOR RESTART
   TRAP UNIT ERROR
   TOO MANY ASSIGN CARDS
   TWO RUN CARDS
   UNIT ASSIGNED TWICE

Units 6 and 57 must be assigned only to the dd80. Seven or less units may be assigned.
IOCHEK

Called by IF UNIT tests in Fortran statements.

IF UNIT CALL BEFORE I/O CALL

In a buffer operation on a given unit, the IF UNIT tests may not
be used before the first buffer statement.

IOPROC

LOGICAL TAPE NO. MUST BE 1 THROUGH 10

The tape unit must be in the range 1-10 or the program will terminate.

LENGTHF

LENGTHF CALL BEFORE I/O CALL

The buffer I/O statement on unit i must be programmed before
calling the LENGTHF(i). Check the logical unit numbers.

LOADER

ATTEMPT TO REDEFINE COMMON BLOCK LENGTH

Examine the size of the arrays in common. A subroutine contains more
storage in a particular common block than the first reference to this
block. (Binary DECKS are loaded first. This could be because new
binary decks are used, and thus loaded ahead of a MAIN program.)

BLANK COMMON MAY NOT BE PRESENT

Remove data statements which assign values to variables in blank common.

CARD MISSING OR DECK OUT OF ORDER

Binary cards are probably out of order. If no binary cards are
 included in this job, see a systems programmer.

CHECKSUM ERROR

The checksum recalculated does not agree with the original checksum.
Try running the program again. You may have to remake binary cards.
There may be a hardware or systems software error. See a systems
programmer.

ENTRY POINT TABLE OVERFLOW

There are more than 250 entry points specified in the program (job).

EXTERNAL AND ENTRY TABLES DON'T MATCH

There is a missing subroutine the program calls which does not exist
in the library. There will be a list of missing subroutine names.
Look for variables which need to be dimensioned or names with a zero
instead of an 0.

EXTERNAL CHAIN TABLE OVERFLOW

There can be no more than 200 unique externals referenced in
a program (job).
FULL DRUM, RESUBMIT
There is no place on drum to store OVERLAY.

LIBRARY DIRECTORY ERROR
There is an error in the system tape library directory.
See a systems programmer.

LIBRARY EXTERNAL REQUEST OVERFLOW
The number of unique external symbols must not exceed 100 for library routines requested by the program (job).

LIBRARY SUBROUTINE REQUEST OVERFLOW
More than 60 library routines have been requested.

LOCAL EXTERNAL TABLE OVERFLOW
No more than 50 external symbols may be referenced in a subroutine.

MAX CORE = 160000 CORE NEEDED =
The program and subroutines have used more core than is available.
Economize on array size by decreasing dimensions, using equivalence or using common more efficiently. You may overlap the loader with a blank common block at least 40008 words long specified in the first program. This may save between 30008 and 40008 locations.

OVERLAY HAS MISSING OR UNDEFINED ENTRY POINT
Examine labels for segments and overlays.

SEGMENTS MISNAMED
Examine segment labels in overlay calls.

TOO MANY COMMON BLOCKS
There are more than 20 labeled common blocks in one routine, or more than 50 in the program (job).

TOO MANY OVERLAYS
Maximum overlay/segment count is 16.

2 ENTRY POINTS WITH THE SAME NAME
Each entry point name must be different from the subroutine name or other entry point names. The substitution of a subroutine for a library subroutine that has external references will give this comment.
OUTPTC/S

OUTPTC is the routine that is called whenever you have a PRINT, PUNCH, WRITE OUTPUT TAPE i,n,L or WRITE(i,n)L. OUTPTS is called whenever you have an ENCODE(c,m,V)L statement. The format statement is not scanned during compilation; it is scanned at execution time.

FIELD WIDTH LESS THAN DECIMAL WIDTH
The field specified requests more digits after the decimal than the width. Example: F6.7 is a field width error.

HOLLERITH FORMAT WITH LIST
No field specification for the list is contained in the format statement.

ILLEGAL FUNCTIONAL LETTER IN FORMAT STATEMENT
A number specification has been made using a character other than X,I,A,R,E,D,F,O,H,*,L,P,+ or -. Look for keypunch errors. A wild index or too small dimension statement may have caused some of core to be overwritten. Also check the count in an H specification.

OUTPUT RECORD LENGTH EXCEEDED
The number of characters the program may write per line is 12. Examine the format statement for too many repeated groups or too many alphanumeric characters without requesting a new record (specified by a slash (/)).

PAREN GROUP NOT CLOSED IN FORMAT STATEMENT
Check paren count; left ( should = # of ) parens.

TAPE ERROR ON OUTPUT FROM LOC
This may be caused by a parity error in the tape unit or you may be trying to write a binary tape under a BCD write.

ZERO FIELD WIDTH IN FORMAT STATEMENT
The w in the format specification is zero, so that you are specifying F0.6 or F0.10, etc. Look for a wild index or dimensions that are too small; the format statement may be clobbered. Check for keypunch error.

OVERLAY

CALL TO OVERLAY OR SEGMENT NOT LOADED
The program has called for an overlay or a segment that has not been loaded with the program deck. A CALL OVERLAY (I0,IS) statement is incorrect. An OVERLAY card has not been placed in a deck. Check description of routine OVERLAY.
PLOT

MORE THAN 25000 PLOT INS ASSIGN DD80=59
Normally generated plotter instructions are stored on the drum until your program is in output phase. Your program has generated a huge number of plotter instructions, normally a program error. If you really mean it, add a control card: *ASSIGN,DD80=59. This will run the dd80 on line (like a tape drive) and bypass the drum storage of plotter instructions.

X OR Y COOR ZERO OR NEG WITH LOG SCALING
Examine plotter coordinates

PM

FLOATING POINT OVERFLOW
A number greater than 10^{322} has been generated. Look for a divide by a very small number or zero, or a function which is increasing exponentially and exceeds this number.

INDEFINITE OPERAND
A ± 1777 in the exponent field is stored in a variable to which the program is applying a floating point operation (/,*,+,-) code. This indefinite has been generated previously somewhere in the program. Look for errors in mixed mode references, or mixed mode equivalence logic, as well as a % operation. Some subroutines pack an indefinite for an error exit.

MEMORY BOUNDS ERROR
The program has referenced an operand or instruction which is outside the core area (FL) reserved for the program. The program may have a "wild" jump and be trying to execute the data. Check array dimensions and DO loop indices. There may be too few arguments in a calling sequence.

OPERATOR DROP
The computer operator dropped the program at the console. A note accompanies the output explaining why he did this. Often a tight infinite loop is the reason. Check the control statement logic (IF's and DO's). Examine the P counter on the panel printout to determine where in the program the drop occurred. This may help find an infinite loop.

PROGRAM DESTROYED LOW CORE
Illegal call to core cell one.

PROGRAM STOPPED AT
A program stop instruction was encountered by the peripheral monitor in testing the first 6 bits (operations code) of the word. From a Fortran program, look for arrays not dimensioned for enough storage, or faulty indexing. The program may be trying to execute data. There may be no external statement for a routine in a parameter list. There may be too few arguments in a calling sequence. If the program has overlap with dimensions and no entry point is provided the program stops.
TIME LIMIT EXCEEDED
The *LIMIT card is missing and the time is assumed to be one minute or the program exceeds the time limit specified on the *LIMIT card.

SAVE

CHECKSUM ERROR
The tape made by the SAVEF routine cannot be read on restart. Check that the correct tape is assigned.

DRUM WRITE OF OVERLAY FAILED
The program is using subroutine OVERLAY. In the process of moving overlays to the drum during the SAVEF procedure, an unrecoverable drum write error has occurred. Notify the operator.

DRUM READ OF OVERLAY FAILED
The same problem as above except that the drum failed on a read operation.

SAVE TAPE READ ERROR
In the process of reading the SAVE tape during a restart there was an unrecoverable read error. Have the operator mount the tape on a different unit and try again.

SAVE TAPE WRITE ERROR
In the process of writing the SAVE tape, an unrecoverable write error has occurred. Assign a new tape to be used as the save tape.

SW6 TERM BY SAVE (no panel is output)
The program is using the SAVEF routine to implement restart procedures on a long job. The operator has set sense switch 6 to force the SAVEF routine to terminate the program for restart at a later time. (This message is printed by SAVEF and not with a RJ SYSERR.)

SET

ARG 5-8 ZERO OR NEG WITH LOG SCALING
Arguments 5 to 8 are the minimum and maximum values to be scaled. With log scaling they must be greater than zero.

TD

REJECT ON TAPE
The peripheral processor was unable to read or write a tape. Two tapes are dialed to the same number or an operator dialed through a tape number. This is a possible machine error; please notify machine room.