PROGRAMMING THE SPS CONSOLES

Paul S. Anderssen

ABSTRACT

The SPS particle accelerator is operated from the control consoles in the main control room. The consoles are equipped with displays and special devices which are programmed to interface the operators to the accelerator equipment via the computer communication network.

This manual describes how the displays and the special devices can be used and in detail how to program them. Programming technique is illustrated with examples throughout the text.
<table>
<thead>
<tr>
<th>Subject</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Console description</td>
<td>2</td>
</tr>
<tr>
<td>3 Console system software</td>
<td>3</td>
</tr>
<tr>
<td>4 The console keyboard</td>
<td>3</td>
</tr>
<tr>
<td>5 Logging in</td>
<td>3</td>
</tr>
<tr>
<td>6 The display system</td>
<td>4</td>
</tr>
<tr>
<td>6.1 Display hardware</td>
<td>5</td>
</tr>
<tr>
<td>6.2 Programming the displays</td>
<td>5</td>
</tr>
<tr>
<td>6.2.1 How to select a display device in the console</td>
<td>6</td>
</tr>
<tr>
<td>6.2.2 How to select a display device from the display pool</td>
<td>7</td>
</tr>
<tr>
<td>6.2.3 Positioning the display on the screen</td>
<td>8</td>
</tr>
<tr>
<td>6.2.3.1 The graphics coordinate system</td>
<td>8</td>
</tr>
<tr>
<td>6.2.3.2 The character coordinate system</td>
<td>9</td>
</tr>
<tr>
<td>6.2.4 Colour displays</td>
<td>9</td>
</tr>
<tr>
<td>6.2.5 Elementary graphics functions</td>
<td>9</td>
</tr>
<tr>
<td>6.2.6 Array oriented graphics</td>
<td>12</td>
</tr>
<tr>
<td>6.2.7 Character mosaic displays</td>
<td>16</td>
</tr>
<tr>
<td>6.2.8 Display layout</td>
<td>18</td>
</tr>
<tr>
<td>7 The video switch system</td>
<td>18</td>
</tr>
<tr>
<td>8 The buttons</td>
<td>19</td>
</tr>
<tr>
<td>8.1 System buttons</td>
<td>19</td>
</tr>
<tr>
<td>8.2 The programmable buttons</td>
<td>19</td>
</tr>
<tr>
<td>8.2.1 Writing the touch button legends</td>
<td>20</td>
</tr>
<tr>
<td>8.2.2 Programming the buttons</td>
<td>22</td>
</tr>
<tr>
<td>9 The knob</td>
<td>22</td>
</tr>
<tr>
<td>9.1 The knob in switch mode</td>
<td>23</td>
</tr>
<tr>
<td>9.2 The knob in spring mode</td>
<td>23</td>
</tr>
<tr>
<td>9.3 The knob in potentiometer mode</td>
<td>23</td>
</tr>
<tr>
<td>10 The tracker-ball</td>
<td>23</td>
</tr>
<tr>
<td>11 The cursor</td>
<td>24</td>
</tr>
<tr>
<td>11.1 Cursor control</td>
<td>24</td>
</tr>
<tr>
<td>11.2 Programming with the cursor</td>
<td>25</td>
</tr>
<tr>
<td>13 The program tree</td>
<td>26</td>
</tr>
</tbody>
</table>
13.1 The trunk of the program tree ...................... 27
13.2 The stack of programs ............................. 27
13.3 The software functions operating on the program tree .. 28
13.4 The standard branch program of the tree (node) ........ 28

14 On-line information retrieval .......................... 31
14.1 The HELP facility for branch programs ............... 31

15 Accelerator timing facilities .......................... 31
16 The local file BEEF ................................ 32

17 Restrictions on programs ............................. 33

References: .............................................. 35

Appendix I. Software functions .......................... 36
Appendix II. Character codes ............................. 62
Appendix III. Character fonts ............................ 66
Appendix IV. Display layout sheet ....................... 68
1 Introduction.

The SPS control room has been in use since the commissioning of the accelerator ring in 1975. The number of general purpose consoles has been increased from three to five. Four special purpose consoles (touch terminals) have also been added to offer more access points to the control system.

The consoles have been subject to certain modifications during the years of operation. The major improvement has been the introduction of the microprocessor driven display system in 1978.

The steady improvement of the user facilities has gradually made obsolete the original documentation on how to write programs for running in the consoles. Much of the old documentation was written with a view to the construction and commissioning phase of the SPS. During the first years of operation the emphasis of the control programs has shifted from basic equipment control towards higher level accelerator operation. At the same time the user community has changed. This manual is intended to replace all old notes with a comprehensive description for today's user of the control room.
2 Console description.

The picture below shows the current layout of the different console components:

Figure 2.1. Hardware layout of the consoles.

The original design and construction of the consoles are described in [6].

The hardware components of the consoles are connected to the console computer by means of a CAMAC interface [2].
3 Console system software.

Efficient operation of the SPS relies heavily on the quality of the application programs. In particular the way information is presented to the operator and his possible means of interaction with the control programs are of vital importance. Therefore it is essential to encourage extensive and logical use of the special hardware in the consoles. A large number of built in software functions makes the programming simple and straightforward even for the non-specialist programmer. These functions can be regarded as an extension to the NODAL interpreter which is the basic programming language in the control system [1]. Working knowledge of NODAL is a prerequisite for a full understanding of this manual.

4 The console keyboard.

The consoles are equipped with a keyboard which can be used to edit NODAL programs and to enter data into programs. The NODAL manual [1] describes the details of these operations. In addition the keyboard can be used to log in to the control system. This is described in the next section.

The console keyboard is the default input device (IDEV) for the interactive NODAL in the consoles. Application programs are most conveniently typed in and tested on the consoles themselves, using the actual hardware for debugging the programs. Note that NODAL is an interpretive language which does not require compilation nor link editing of the program modules so that the effect of the programmer's efforts are immediately displayed.

5 Logging in.

In order to become an approved user of the control system a new user should contact the author (Paul S. Anderssen) who maintains the list of users and the associated password files. Users will be removed from the password file when they leave CERN.

The first thing a user has to do when he wants to work at a console is to log in. Files and equipment are protected so that no unauthorized user can disturb the operation of the accelerator. By logging in the user makes himself known to the system. The system software looks him up in the password file and establishes his authority according to his entry there. The competence of a user is determined by two entities: his section number and his capability. The former will generally give him authority to change programs and static data tables. This is related to his area of work. The latter will give him power to access the data modules which control the equipment. This is more related to his capabilities as an accelerator operator. The
password and protection system is described further in [11].

There are two ways to log in: either by inserting the SPS identity badge into the badge reader on the console and pressing the RESET button, or by typing the following command on the keyboard:

>SET USER=<CERN identity number of user>

The system will respond with the prompt:

PASSWORD:

Now the user must type his password which will not be echoed.

To terminate work at the console the user is advised to log out. This is done by pressing RESET with the badge removed or by typing:

>SET USER=0

Once logged in to the control system as an individual, the user may change his section by typing:

>SET SECTN=<the section number>

The system will again respond with the prompt:

PASSWORD:

Here the user must type the section's password which will not be echoed. Note that changing section in this way does not change the user's capability but gives him access to files and data modules in different sections as long as the system has been informed of a multi section capability.

6 The display system.

The control room is equipped with television display screens which can be programmed to give visual feedback to the operators. Each console has six such screens mounted in the display surface (figure 2.1).

Pictures can be generated by computer programs running in the console computers, the news computer, and in the display computer. The present section describes how such computer generated pictures are created on the available hardware. How to connect other TV picture sources to the console screens is described in section 7.
6.1 Display hardware.

Conventional European television pictures are used throughout the system [3]. This standard specifies 525 scan lines per picture frame. Only 576 lines are actually visible on the surface of the display screen. Furthermore, the standard defines the aspect ratio of the screen to be three by four. By consequence, the number of picture elements (pixels) horizontally along each scan line becomes:

\[ 576 \times \frac{4}{3} = 768 \]

The total number of pixels required for a full resolution picture is therefore \( 576 \times 768 = 442368 \). A TV raster refresh memory (DIME) contains a storage element for each pixel. For economy reasons the DIMEs which are installed at present contain only a quarter of the total number of storage elements in the memory. For the same reason only monochrome DIMEs have been installed even on the colour screen. However, multicolour pictures can be obtained by connecting one DIME to each of the three colour channels (red, green and blue). Only the red and green channels are used in this way in the consoles.

A microprocessor driven graphics controller (DICO) is attached to each DIME. The firmware of the DICO translates high level graphics commands into pixel information.

In addition to the graphics, less costly character displays are in use in the control room. They are normally used for displays where general purpose graphics is not essential. There is a monochrome character display under the touch buttons and there is a four colour character display on the central colour screen.

6.2 Programming the displays.

In spite of the different types of display hardware, the application programs see only one interface to the various displays. Similar capabilities, like the display of a text, are addressed in a uniform manner regardless of whether the actual display device is a character display or a graphics display. Naturally, proper graphics are only possible on the DICO/DIMEs.

Numerous built-in software functions are available for drawing on the displays (appendix I). The very first action taken by a display program is to select the display device to use. This is done by setting the output device (ODEV) in NODAL. Subsequent calls to the display functions will then act on the selected device.
The display devices can be programmed on three levels from an application program: by submitting characters (ASCII codes), by calling the basic graphic functions, or by calling the array oriented display functions. They are capable of generating complete graphics displays of data stored in arrays with only a few subroutine calls.

6.2.1 How to select a display device in the console.

Fixed numbers are assigned to each display device as follows:

On the colour screen:

10 = Character mosaic display.

11 = Red half resolution graphics display.

12 = Green half resolution graphics display.

On the large black and white screen:

14 = Half resolution graphics display.

On the touch panel:

30 = Monochrome character display for touch button legends.

(This legend device is normally written by the LEGEND function but can also be used as a normal character display.)

Selection of a display device is made by the ODEV function in NODAL.

For example:

>SET ODEV=12; TYPE \12

selects the green channel on the central colour screen. In addition the ASCII code number 12 (form feed) is output to that channel. Form feed is particularly useful to reset a display device. The foreground colour is set to white and the background colour to black. The case shift option is turned off, the normal character set is chosen, and the writing direction is made normal. Finally the screen is cleared and the current position is set to the first character column of the first line. The meaning of this control code as well as the others is described in appendix II.
The ODEV is reset to the default device when the program is terminated. The default device is the on-line terminal screen on the consoles.

6.2.2 How to select a display device from the display pool.

There is no local display device connected directly to the four small screens on the left hand side of the consoles. However, the display computer manages a pool of DICO/DIMES which can be allocated on demand. When a display device is taken out of the pool and assigned to a screen, the video line is automatically switched to the screen. This switching is done in the video switch matrix which is discussed in section 7. The display device is driven by the display computer. Hence, such display programs must be executed in that computer. The argument of the ODEV statement is the screen number of the console in the range 15 to 18.

An example illustrates this programming technique:

```
1.10 EXECUTE(DISp) 10; WAIT(DISP)
1.90 END

10.10 SET ODEV=15; % SELECT TOP LEFT SCREEN, GET FREE DISPLAY FROM
10.12 % POOL IF NECESSARY AND CONNECT IT TO SCREEN
10.20 TYPE \\12; % RESET AND CLEAR THE SCREEN
10.30 TYPE DAYTIM; % DISPLAY DAY AND TIME ON TOP LINE
```

The number of display devices in the pool is limited to eight. If all are allocated, the error message "DEVICE NOT CONNECTED" is returned from the ODEV command.

It is possible to disconnect all displays from a screen by means of the statement:

```
>IMEX(DISp) BREAK(15)
```

This will disconnect everything from screen 15 (top left hand small screen). Any reserved DICO/DIMES will be liberated and returned to the pool.

The program (LIB)<1>SWITCH which is run by pressing the SWITCH button on the console, has a touch button page which allows the operator to control the DICO/DIMES in the pool. There is one button, annotated "WHERE ARE THE DICO/DIMES" which, when pressed, shows which screens the reserved DICO/DIMES are allocated to. Another button allows the operator to liberate one if so desired. However, if the liberated display was used by a scheduled program, the program will
try to reserve another the next time it runs.

Note that each subprogram which is submitted to the display computer must contain the setting of ODEV. However, when the display computer finds that a display device is already allocated to that screen, then no reallocation is made. Thus successive executes can change an existing display.

6.2.3 Positioning the display on the screen.

Each display device has a store for the "current position" of the display. This is the base position for many of the elementary display functions. Calls of display functions may change the current position as specified for the function (appendix I). An output character will be displayed at the current position and the new current position will be the next character column.

Graphics and text call for different ways of addressing the screens. Graphics require that each pixel is individually addressable. Text displays are normally positioned in terms of lines and columns. In the DICO the two positioning methods coexist and can be used alternately, but note that only a single memory element is available to store the current position.

6.2.3.1 The graphics coordinate system.

The full resolution graphics hardware offers theoretically 768 by 576 pixel positions on the screen. This mesh is used for programming the displays. It is even used for programming the half resolution DIME. In this case the firmware of the DICO takes care of the necessary scaling. Hence, the user program can be transported from one type of DIME to another without modification.

The pixel in the bottom left hand corner is addressed as [0,0] and the one in the top right hand corner as [767,575]. It is possible to displace the origin by means of the ORIGIN function:

>SET ODEV=14; ORIGIN(384,288)

These commands will move the origin on the large monochrome screen to the center of the screen surface. The origin is moved back to the bottom left hand corner by:

>SET ODEV=14; ORIGIN(0,0)
It is possible to move the "current position" to any address on the screen by the function MOVE:

```plaintext
>SET ODEV=14; MOVE(100,200)
```

These commands will move the "current position" to the pixel whose coordinates correspond to [100,200] relative to the present origin.

6.2.3.2 The character coordinate system.

When drawing character displays, one might prefer to position the display in terms of lines and columns. The standard character size yields 24 lines of 64 columns each. Line 1 being the top line, and line 24 being the bottom line. The columns are numbered from left to right, 1 to 64.

6.2.4 Colour displays.

It is possible to create colour displays on the central colour screen of the consoles. Two monochrome DIME have been connected to the red and green channels of this screen. This allows colour graphics to be created. The two channels must be programmed as individual display devices. They have the ODEV numbers 11 (red) and 12 (green).

In addition there is a colour character generator (made by Christian Rovsing) connected to this screen. This is a true multicolour device which obeys the colour control commands listed in appendix II.

6.2.5 Elementary graphics functions.

The elementary graphics functions are part of the standard NODAL function library. The functions ORIGIN and MOVE which deal with positioning on the screen belong to this class of functions. The programmer can talk to the display system in a simple language. To draw a straight line segment from the current position to the point [X,Y] requires only a simple program statement VECT(X,Y). A rectangular box is drawn by BOX(X,Y) and a filled box by SOLID(X,Y). These two rectangular forms are drawn in such a way that the straight line through the current position and [X,Y] lies on their diagonal. A circle of radius R can be drawn by the statement CIRCLE(R). The circle is centered around the current position. Parameters to these functions are always in screen graphics coordinates. Appendix I gives a description of all the currently available functions of this class.
The following example shows how a display of a mathematical function can be programmed using the basic graphics functions:

1.02 % DEMONSTRATION PROGRAM TO DISPLAY AN ELEMENTARY MATH. FUNCTION
1.10 SET M=50; % NUMBER OF SAMPLES
1.20 SET W=2*PI

2.02 % SET BASE LINES FOR DISPLAY
2.10 SET BX=100
2.20 SET BY=576/6
2.30 SET Y1=BY; SET Y2=3*BY; SET Y3=5*BY

3.02 % SET SCALING FACTORS FOR DISPLAY
3.10 SET SX=BY; SET SY=BY

4.02 % SELECT DISPLAY SCREEN AND DRAW AXIS
4.10 SET ODEV=14; TYPE \12
4.20 SET XE=W*SX+BX
4.30 MOVE(BX,0); VECT(BX,575); MOVE(BX,Y1); VECT(XE,Y1)
4.40 MOVE(BX,Y2); VECT(XE,Y2); MOVE(BX,Y3); VECT(XE,Y3)
4.60 TYPE \11\24 \9\9 "0" \16\8 "0"
4.70 TYPE \9\57 \16\0 "360" \16\8 "0"

5.02 % DRAW SINE WAVE AS CONTINUOUS GRAPH.
5.10 MOVE(BX,Y1); FOR X=W/M,W/M,W; VECT(BX+X*SX,Y1+SIN(X)*SY)

6.02 % DRAW COSINE WAVE AS A COLUMN DIAGRAM
6.10 FOR X=0,W/M,W; MOVE(BX+X*SX,Y2); VECT(BX+X*SX,Y2+COS(X)*SY)

7.02 % DRAW A COMPOSITE FUNCTION AS A SCATTER DIAGRAM
7.10 TYPE \19\16\8; % SELECT SHAPE OF POINT
7.20 FOR X=0,W/M,W; POINT(BX+X*SX,Y3+SIN(X)*COS(X)*SY)

9.90 END
The resulting display will look like this:
6.2.6 Array oriented graphics.

A higher level of graphics functions is to be found in the set of
array oriented display functions. They create displays from data which
are stored in arrays (program data). The programmer does not have to
worry about scaling factors and offsets when converting from program
data to screen graphics coordinates. This is done automatically for
him by the display functions. The program has to specify the area on
the screen where it wants the display to appear with a call of the
SCREEN function. Then it has to specify the range of the program data
values which must fit into the area. This is done by calling the
WINDOW function. Then the program calls one of the display functions
which can draw a complete graph of the whole data set in the form
defined by the function. The different display functions are described
in appendix I.

If one needs to convert from screen coordinates to program data
(or the other way around), one may acquire the scaling factors and
offsets which have been computed internally by the functions SCREEN
and WINDOW. The functions XSCAL and YSCAL return the scaling factors,
and XOFF and YOFF return the offsets. The basic scaling formula is:

screen coordinate:=(physical value-offset)*scaling factor

The following programming example shows how one can use the array
oriented graphics functions to present a Nyquist diagram of the beam
transfer function. The corresponding real and imaginary part of the
function are loaded from a file.

1.02 % PROGRAM TO SHOW ARRAY-ORIENTED GRAPHICS.
1.10 SET ODEV=14; TYPE \12

2.02 % LOAD DATA FROM THE LIBRARY, REAL AND IMAGINARY
2.20 LOAD (LIB)<116> RP
2.30 LOAD (LIB)<116> IP

3.02 % SET DISPLAY PARAMETERS
3.10 SCREEN(0,767,30,575)
3.20 WINDOW(MIN(RP),MAX(RP),MIN(IP),MAX(IP))
3.30 IF VALID<>0; TYPE "ILLEGAL PARAMETERS TO WINDOW" !; END

4.02 % DISPLAY IT
4.10 XYPLT(RP,IP)
4.20 XAXIS; YAXIS

5.02 % LABEL THE PICTURE
5.10 TYPE \11|24 \9\10 "NYQUIST DIAGRAM OF BEAM TRANSFER FUNCTION."

9.90 END
The resulting display will look like this:

In this example, corresponding values of x and y were loaded from two separate arrays, RP and IP. Often one of the components is a monotonically increasing function. It could be the sample number in a repeated data acquisition. In this case, the array index could serve as one component, and the other could be the array element itself. The following sample program shows how this works in case of a histogram type display.
1.02 % Program to display up-time of SPS control computers.

2.02 % Initialize variables
2.10 SET N=55;
2.20 DIMENSION CT(N); % number of computers
2.30 DIMENSION HT(200); % array to store up-times
2.40 SET XL=26; SET XH=640; % array to accumulate histogram
2.50 SET OD=15;
2.60 SET H=60*60*24; % screen bounds for display
2.70 SET ODV=OD; % number of the display device
2.80 SET ODV=OD; % seconds per day

3.02 % Aquire data and display for all computers
3.10 FOR C=1,N; DO 30!32
3.20 SET OD=ODV; TYPE \11\1 \9\1 1\1\1 \9\1 65-SIZE(DAYTIM) DAYTIM
3.30 SET OD=ODV; DO 20
3.40 SET OD=ODV; DO 21

9.90 END

20.10 TYPE \11\1 \9\1 "Days since restart of computer."
20.20 SCREEN(XL,XH,340,550); WINDOW(.5,ARSIZE(CT)+.5,0,MAX(CT))
20.30 HISTO(CT)
20.90 YAXIS; XAXIS; TYPE \9\56 "Comp. no."

21.10 TYPE \11\14 \9\1 "Distribution in time since restart."
21.20 SCREEN(XL,XH,24,230); WINDOW(.5,MAX(CT)+.5,0,MAX(HT)*8/7)
21.30 DIAGRM(HT,HORIZO)
21.90 YAXIS; XAXIS; TYPE \9\56 "days"

30.02 % Subprogram to process data from one computer
30.10 IF DLSTAT(C)=0; RETURN; % computer-link open?
30.20 EXECUTE(C) 31; WAIT(C); % acquire uptime
30.30 SET T=T/H;
30.40 SET CT(T)=T; % convert to days
30.50 SET HT(T+.5)=HT(T+.5)+1; % store Computer Time
30.60 SET HT(T+.5)=HT(T+.5)+1; % accumulate time bin

31.02 % Subprogram executed in remote computer
31.10 SET T=TIME
31.20 REMIT T

32.02 % Subprogram to catch and report errors in acquisition
32.10 TYPE \1 ERME(ER) " at line " %5.02 ERRLIN
32.20 TYPE "in computer no." %3 C
The display will look like this:

Days since restart of computer.

Distribution in time since restart.

Note that the HISTO function is used for the upper display and that the DIAGRM function is used for the other.
6.2.7 Character mosaic displays.

All the display devices contain a character font which includes special characters for building up mosaic pictures. The available font is shown in appendix III. Such pictures can be displayed on both the DICO and the character devices without program modification. The visual properties of mosaic pictures can be quite acceptable for mimic diagrams but, because of the crude resolution, it results in inferior graphics. The following programming example shows how the special character font may be used to frame in tabular data:

1.02 % PROGRAM TO DRAW A TABLE OF SIN(X)
1.20 SETODEV=14; TYPE \12
1.30 TYPE LINE(10) COLUMN(20) "TABULAR DISPLAY USING"
1.40 TYPE LINE(11) COLUMN(20) "SPECIAL MIMIC CHARACTERS."

2.10 % GROUP 2 DRAWS FRAMEWORK AROUND THE TABLE
2.20 TYPE LINE(2) COLUMN(1)
2.30 DO 10; TYPE \133 \13 \130 \25 \25 \25 \139
2.40 TYPE ! \128 " X " \128 " SIN(X) " \128
2.50 DO 10; TYPE \137 \13 \136 \25 \25 \25 \140

3.10 % GROUP 3 PRINTS THE VALUES
3.20 SET M=12; % NUMBER OF VALUES
3.30 FOR X=0,2*PIE/M,2*PIE; DO 30

4.10 % GROUP 4 FINISHES FRAMEWORK
4.20 DO 10; TYPE \131 \13 \132 \25 \25 \25 \138

9.99 END

10.02 % HORIZONTAL BARS
10.20 TYPE !; FOR J=1,13; TYPE \129

30.02 % PRINT A LINE WITH VALUES
30.10 TYPE ! \128 %3 360*X/2*PIE \128 %8.03 SIN(X) \128
By running the program the display will look like this:

<table>
<thead>
<tr>
<th>X</th>
<th>SIN(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>.5</td>
</tr>
<tr>
<td>60</td>
<td>.866</td>
</tr>
<tr>
<td>90</td>
<td>1</td>
</tr>
<tr>
<td>120</td>
<td>.866</td>
</tr>
<tr>
<td>150</td>
<td>.5</td>
</tr>
<tr>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>210</td>
<td>-.5</td>
</tr>
<tr>
<td>240</td>
<td>-.866</td>
</tr>
<tr>
<td>270</td>
<td>-1</td>
</tr>
<tr>
<td>300</td>
<td>-.866</td>
</tr>
<tr>
<td>330</td>
<td>-.5</td>
</tr>
<tr>
<td>360</td>
<td>0</td>
</tr>
</tbody>
</table>

This example shows another disadvantage of the mosaic display: the program is not easy to read and hence difficult to maintain. To overcome this problem an interactive suit of programs has been written which helps the user to build up his mosaic picture using the interactive devices of the consoles. This program is called (LIB)<96>BUILD and it automatically enters data into an array in response to user input. The array can be stored in a file for
subsequent retrieval and display. A special display function PINXIT
decodes the data in the array and outputs the picture on the current
ODEV. The description of the PINXIT function is contained in appendix
I. The background picture of PAGE1 is such a display. (PAGE1 is the
main display of the current SPS accelerator status.)

6.2.8 Display layout.

Complete displays are generally composed of both text and
graphics. To help the programmer to plan his display a special
display layout sheet has been prepared. The line/column positions are
shown as well as the graphics coordinate system. A specimen is
attached in appendix IV.

7 The video switch system.

TV pictures can be switched to the four small screens on the
consoles as well as to other screens on both CERN sites. The switching
of allocatable DICO/DIMEs is also done in this way. The video switch
matrix is controlled by the display computer. A set of display
computer functions helps the programmer to manipulate the matrix
connections. Of particular importance for console programs are the
functions APPEAR and BREAK. APPEAR switches a TV picture to a screen.
Two parameters have to be supplied with the call: the input number in
the video switch matrix and the screen number of the console. The
input numbers are shown on the drawing CERN-SPS 5.2150.01.001.0. The
function BREAK will disconnect all TV sources from a console screen.
These functions will automatically take the console number into
account when they work out the switch number in the matrix. The
following example shows how to program the switch:

1.10 WHILE 0=0; DO 3; % do line-group 3 repeatedly
1.90 END

3.02 TYPE ! "Program to connect a TV picture to a console screen." !
3.10 ASK "Switch matrix input (ref. CERN-SPS 5.2150.01.001.0)" IN
3.20 IF IN>87 OR IN<8; TYPE "Illegal input, try again!" !; GOTO 3.1
3.30 ASK "Console screen number, 15 to 18" SN
3.40 IF SN>18 OR SN<15; TYPE "Illegal screen, try again!" !; GOTO 3.3
3.50 EXECUTE(DISP) 10 IN SN; WAIT(DISP)
3.90 RETURN

10.10 BREAK(SN); % disconnect possible old pictures
10.20 APPEAR(IN,SN); % connect the demanded picture
10.30 RETURN
The interactive program (LIB)<1 SWITCH helps the operators to connect the standard displays to the more common screens. The program is started by pressing the SWITCH button. It uses the touch buttons to select the desired picture and the destination screen.

8 The buttons.

The consoles are equipped with several buttons. Some serve special system purposes, while others are available for the application programs. System buttons are all mechanical push buttons such as TRUNK, see figure 2.1. The programmable buttons can be either mechanical buttons or touch buttons.

8.1 System buttons.

The system buttons are the four push buttons beneath the central colour screen and the BACK button adjacent to the touch panel. These buttons cannot be accessed from the programs in the normal way, but they serve important system purposes.

The TRUNK and BACK buttons are explained in conjunction with the console program tree.

The HELP button is discussed in the section 'On-line information retrieval'.

The SWITCH button is discussed in conjunction with the video switch system.

The purpose of the RESET button has been discussed briefly in the section 'Logging in'. In addition it disconnects and aborts all programs in the console and restarts the interactive NODAL with an empty working area.

8.2 The programmable buttons.

The most important of the programmable buttons are the touch buttons. The touch buttons consist of a display screen covered with a touch sensitive surface. There are 16 touch sensitive button areas on this surface. It is possible to display a text under each button.

These are used on the console as the primary device for organising and selecting all the SPS operational programs. Very few programs are called from the console by direct entry via the keyboard.
In addition there are several mechanical push buttons which are accessible from the user programs. All the programmable buttons which are functionally related are assembled logically into button groups. The button group number, supplied as a parameter to the call of the button functions, allows the console software to distinguish between the various groups i.e.

>SET Z=BUTTON(5)

The default is group number one. The button groups are:

1 = Main touch buttons (default).
2 = Spare button group
3 = Button next to the ball.
4 = Button next to the knob.
5 = Buttons to identify display screens.

8.2.1 Writing the touch button legends.

Each touch button is identified as being active by a text or legend under the sensitive area. The legend can contain three lines of text, each ten characters long. The software function LEGEND does this. The parameter to LEGEND is the button number: 1 to 16, numbered from left to right and from top to bottom. The parameter value zero writes a centered single line heading on the touch button page.

10.10 $SET LEGEND(0)="THIS IS THE HEADING"
10.20 $SET LEGEND(1)="THE FIRST\BUTTON"
10.30 $SET LEGEND(6)="THIS IS\BUTTON\NO.6"
10.40 $SET LEGEND(11)="THIS IS\BUTTON\NO.11"
10.50 $SET LEGEND(16)="THE LAST\BUTTON"
The resulting touch button display looks like this:

```
THE FIRST BUTTON
```

```
THIS IS BUTTON NO.6
```

```
THIS IS BUTTON NO.11
```

```
THE LAST BUTTON
```

Note that the '\' (back slash) moves the text to the next line in the button legend.

It is very convenient to build up suites of control programs around the touch button concept. Recommended practice for doing so is presented in the section 13 'The program tree'.
8.2.2 Programming the buttons.

Buttons are used in two ways by interactive programs. In the first mode the program waits for a button to be pressed. In the second mode it tests occasionally for a button being pressed while executing another part of the program.

The console software supports both these programming techniques by means of the two button functions BUTS and BUTTON. BUTS returns to the program immediately, with a zero if no button was pressed otherwise with the number of the button. BUTTON waits until a button is pressed. Then it returns with the button number.

The following programming example shows how BUTS works:

10.10 $SET LEGEND(0)="BUTS TEST"
10.20 $SET LEGEND(11)="\TEST\ BUTTON"
10.30 WHILE BUTS=0; TYPE \13 "NO BUTTON HAS BEEN PRESSED YET!"
10.40 TYPE ! "NOW A BUTTON HAS BEEN PRESSED!"

The following example shows how BUTTON works:

11.10 $SET LEGEND(0)="BUTTON TEST"
11.20 $SET LEGEND(11)="\TEST\ BUTTON"
11.30 SET Z=BUTTON; % WAIT FOR BUTTON TO BE PRESSED
11.40 TYPE ! "NOW BUTTON NUMBER:" $3 Z " WAS PRESSED!"

9 The knob.

The knob is a programmable device which allows numbers to be entered into a program. The software function KNOB reads the current value of the knob. This function also allows presetting of the knob value. The knob value increases with clockwise rotation. The knob hardware can be programmed to operate in one out of three modes:

1. Click switch mode.
2. Spring return mode.
3. Potentiometer mode.
The mode is selected by means of the software function MKNOB. This function also allows a brake force to be applied on the knob to give artificial operator feed-back if so desired. The brake force can be set to a value between 0 and 127.

>MKNOB(3,64); % SET TO POTENTIOMETER MODE, APPLY HALF BRAKE FORCE

9.1 The knob in switch mode.

In this mode a built-in clutch engages a ratchet mechanism which makes the knob feel like an electrical rotary switch. It has 16 discrete positions per revolution and about 1800 distinguishable positions in total. Each position is associated with an integer value which is incremented in steps of one from one position to the next. Presetting the switch value is done by:

>SET KNOB=1

And reading the actual switch position is done by:

>SET A=KNOB

9.2 The knob in spring mode.

The knob returns an integer in this mode. The sensitivity is much greater than in the switch mode, 512 counts per revolution compared to 16. The hardware allows about 40 degree deviation from the neutral position, clockwise as well as anticlockwise. When the knob is turned away from the neutral position the spring force tries to pull it back.

9.3 The knob in potentiometer mode.

This is the most used mode of the knob. It returns an integer value in the range from -30000 to +30000 with 512 counts per revolution.

10 The tracker-ball.

The tracker-ball is situated in the table top, to the right of the keyboard. It is most frequently used to steer the cursor on the two large screens. It has a sensitivity and range which is designed for that purpose. The rotation of the ball is detected around two axes perpendicular to each other and parallel with the surface of the
table top. The angular position can be read at any time by means of the two functions HBALL and VBALL referring to the corresponding motion of the cursor: horizontally or vertically. The number range is 0 to 767 horizontally and 0 to 575 vertically, compatible with the addressing scheme of the TV graphics displays. The functions HBALL and VBALL also permit presetting of the position when applied in the write mode.

A facility has been provided to "hook" the cursor to the ball in order to reduce the time lag between ball motion and cursor reaction. This has been implemented as a cursor function and is further described below.

11 The cursor.

The cursor is one of the most important devices for operator interaction. It is used in nearly all the interactive console programs and serves as the operator's index finger to the displays. Usually, a selection of choices is presented on the screen, and the operator has to choose between them by moving the cursor to the item required and then pressing a button. Normally the cursor is steered by the tracker ball and the button next to it is used as the prompting device.

The importance of the cursor in an interactive console has justified extensive software support at the system level. Eight NODAL functions are available altogether: three control the behaviour of the cursor, and five allow reading and writing of the cursor position. The cursor position may be set and acquired both in character and in graphics screen coordinates.

11.1 Cursor control.

The cursor can be connected to the two large screen. It will appear in white on the black and white screen and in yellow on the colour screen. The function CURCON controls the switching of the cursor from one screen to another. It takes the screen number as the only parameter. For example:

> CURCON(10)

will make the cursor appear on the colour screen. To remove the cursor from both screens the parameter value has to be set to zero.

> CURCON(0)
The cursor has several operational modes controlled by the function **MCURS(M,H)**. The first parameter \( M \) selects the form of the cursor display:

- \( M=0,1 \) - suppress cursor
- \( M=2 \) - horizontal line
- \( M=4 \) - vertical line
- \( M=6 \) - cross (vertical and horizontal line)
- \( M=8 \) - rectangular solid

All forms may be programmed to blink by adding one to the mode number.

The second parameter controls the coupling between the tracker ball and the cursor. The cursor movements may be hooked to either of the two axis of rotation of the ball, or both:

- \( H=0 \) - unhook the cursor from the ball
- \( H=1 \) - hook the cursor to horizontal ball movement
- \( H=2 \) - hook the cursor to vertical ball movement
- \( H=3 \) - hook the cursor to ball movement in both directions

For example:

```plaintext
>MCURS(6,3)
```

will display a cross following the ball motion in both \( x \) and \( y \).

After the **MCURS** statement the cursor size is set to its default. The width of the lines is set to one. The width of the solid (for \( M=8,9 \)) is 12 and its height is 20.

It is also possible to program the size of the cursor with the function **CURSIZ(X,Y)**. The first parameter is the width of the cursor display and the second is the height. Both of these parameters are given in standard screen coordinates: width from \( 0 \) to \( 767 \), height from \( 0 \) to \( 575 \). Parts of the cursor display exceeding the screen bounds will be scissored.

### 11.2 Programming with the cursor.

Once the cursor has been steered to the desired position on the screen the operator pushes a button. The program must be able to read the position of the cursor at that instant. Three sets of functions are available to help the programmer to acquire the cursor position in the most convenient units for the application. **CURC** and **CURL** will return the column and line position respectively in the character mesh of 24 lines of 64 characters. On the standard graphics display the function **HCURS** will return the horizontal position in the range \( 0 \) to
767, and VCURS will return the vertical position in the range 0 to 575.

A small programming example will illustrate the standard way to program the cursor:

10.02 \% Example of cursor interaction programmed in NODAL
10.10 MCURS(8,3); \% small square cursor hooked to the ball
10.20 CURSIZ(4,4); \% choose cursor size
10.30 CURCON(10); \% connect cursor to main colour screen
10.40 SET Z=BUTTON(3); \% wait, read button when pressed
10.50 \%
10.52 \% the program will not proceed beyond line 10.40
10.54 \% before the ball button has been pressed
10.56 \%
10.60 SET C=CURC; \% read the column position of the cursor
10.70 SET L=CURL; \% read the line position of the cursor
10.80 MCURS(0,0); \% blank the cursor and unhook it from the ball

The four functions HCURS, VCURS, CURL, and CURC can also be used in the write mode in order to position the cursor to a particular point on the screen. This can be used by the program to draw the operator’s attention to an item on a display.

The cursor can be programmed to give the operator a selection capability. The program presents the different options on the screen and the operator moves the cursor with the tracker ball to approximately where his choice is displayed. He then tells the program that he has made his decision by pressing the ball button. The function CFIND simplifies the programming of this kind of interaction. The programmer sets up a two dimensional integer array filled with the line and column numbers of the character position which represents the "center of gravity" of the display of the option. When this array is supplied as a parameter to CFIND, it will return the index into the array where the character coordinates nearest to the current cursor position are stored.

13 The program tree.

The interactive programs used at the consoles are organized in a hierarchical structure called "the program tree" [12]. Each level of the tree presents the operator with a new set of options to choose from, selectable by touch buttons. The very first level is known as the "trunk" of the tree. All subsequent branches have at least one connecting route from the trunk.
A uniform structure has been established for all normal branch points. There is a file associated with each branch point. This file contains a small program in line-group number two and the two string arrays LEGS and PROGS. It is only when the selection process reaches the "leaf" of the tree that the operator brings to the console the application program which uses the console facilities and acts on the accelerator equipment. The system is only able to keep track of the choices made by the operator if the program selection process conforms with the rules for standard branch programs (see below). This is important for BACK and HELP to work properly. It is also of great help for the Operations group in trying to manage the control programs of the accelerator.

13.1 The trunk of the program tree.

The file (LIB)<1>TRUNK is the highest level branch point. It is the trunk of the program tree. When it is run it presents the operator with a touch panel displaying the basic options for the control of the accelerator. The options may be different for the various modes of the machine. For example in fixed target physics the touch buttons of the trunk differ slightly from the one used in storage mode. The trunk program contains special program statements to choose between the basic modes of the accelerator. It is run when the system button labelled TRUNK is pressed.

13.2 The stack of programs.

As the operator makes choices up the tree, the file names of the branch programs are stored on a stack. This allows easy recapture of preceding branch points for alternative choices. The system button labelled BACK chops one program name off the stack and runs the program file which is at the top of the stack.

The function THISP returns the name of the program at the top of the stack. For example:

>TYPE THISP

responds with:

<1>TRUNK

when the trunk program is at the top. The computer number will be added if the computer where the file is stored is not the library, i.e.:

>TYPE THISP
(36)<8>RAP.

will be the reply when the program (RP)<8>RAP. from the radiation protection branch of the tree is running on the consoles in the main control room.

The trunk program sits permanently in the bottom of the stack. When the TRUNK button is pressed the stack is cleared and only the trunk program itself remains. Thus consecutive activations of the BACK button beyond the "TRUNK" level has the sole net effect that the TRUNK program is run again.

13.3 The software functions operating on the program tree.

The software of the consoles support the concept of the program tree with four user functions: TRUNK, NEXT, BACK and THISP. The TRUNK function clears the program stack. The NEXT function takes a file name as a parameter. It pushes the name onto the stack and runs the program from the lowest line number present, as does the RUN command in NODAL. The BACK function pops a name off the stack and runs the program name which popped up, starting from the lowest line number in line-group number two. The different choice of starting line between NEXT and BACK allows the applications programmer to place NODAL commands in line-group one which will not execute when the program is run by BACK. It can be a statement to load defined functions which are commonly used by subsequent branches in the tree.

The function THISP without any parameters returns the name of the program on the top of the stack. An optional parameter allows spying on the contents deeper down in the stack. The value of the parameter gives the level, counting from the top, with zero being the top level.

13.4 The standard branch program of the tree (node).

A standard branch program is listed below. The template is stored in (LIB)<1>BRANCH.

2.02 % Standard branch program (Node)
2.20 SET I=BUTS; FOR I=1,17; $SET LEGEND(I-1)=LEGS(I)
2.30 SET I=BUTTON; $SET LEGEND(0)=; SET ARG(1)=I
2.40 IF SIZE(PROGS(I))>0; $DO PROGS(1)
2.50 $SET S=LEGS(I+1); $MATCH S"=" " ; $MATCH S"=" "
2.60 $SET LEGEND(0)=S &2 PROGS(I+1); NEXT(PROGS(I+1))

LEGS 56 STRING ARRAY
PROGS 104 STRING ARRAY
TX 6 STRING ARRAY
152 WORDS OF TEXT, 166 WORDS OF DATA : LEAVING 19665
The string array LEGS contains the text to display under the touch buttons as legends and PROGS contains the names of the corresponding programs to run when a touch button has been touched. The string array TX contains the text to display on the large monochrome screen when the HELP button has been pressed (see below). The interactive program (LIB)<1>UPNODE allows the programmer to change the legends to be displayed under the buttons (LEGS) and the corresponding action programs (PROGS). It also allows creation of a node from scratch. (LIB)<1>UPNODE will first of all ask for the name of the branch program to update. If no program name is entered at this stage the standard branch program (LIB)<1>BRANCH will be used as the template and the user will be asked to specify the target file name of his branch before saving it at the end of the session.

Please note the special significance of the string PROGS(1) which allows for special application dependent actions in the standard branch program (line 2.4 in the example program above).

The following example shows the contents of LEGS and PROGS from a typical branch in the SPS operations tree. A hard copy of the corresponding touch button display is also shown.

LEGS( 1)=GENERAL SERVICES, <499>BRSERV
LEGS( 2)=COOLING
LEGS( 3)=ELECTRIC\POWER
LEGS( 4)=VENTILATION
LEGS( 5)=VACUUM
LEGS( 6)=RADIATION
LEGS( 7)=SWITCH\SYSTEM\MAINTN.
LEGS( 8)=AUXPS\GEF\TESTS
LEGS( 9)=BEAM\INSTRUMENT\TESTS
LEGS(10)=TUNNEL\LIGHTING
LEGS(11)=
LEGS(12)=\SEM\TESTS
LEGS(13)=
LEGS(14)=
LEGS(15)=
LEGS(16)=
LEGS(17)=GENERAL\SERVICE\ALARMS

PROGS( 1)=
PROGS( 2)<223>COOL
PROGS( 3)<207>MT:1
PROGS( 4)<185>VE:STA
PROGS( 5)<75>VPT:1
PROGS( 6)<8>RADIAT
PROGS( 7)<23>PRESWI
PROGS( 8)<318>PSPACE
PROGS( 9)<66>BMTEST
PROGS(10)<447>FR:LIG
PROGS(11)=
PROGS(12)<323>SEMTP
PROGS(13)=
PROGS(14)=
PROGS(15)=
PROGS(16)<20>SURV
PROGS(17)<207>AL:DIS
The corresponding touch button display becomes:

- COOLING
- ELECTRIC POWER
- VENTILATION
- VACUUM
- RADIATION
- SWITCH SYSTEM MAINT.
- AUXPS & GEF TESTS
- BEAM INSTRUMENT TESTS
- TUNNEL LIGHTING
- SEM TESTS
- GENERAL SERVICE ALARMS
14 On-line information retrieval.

Information about the control system is provided in several ways. It exists both in printed form and stored on various data files in the library. The consoles themselves serve as the most convenient access point to the on-line information. The system button labelled HELP activates the information retrieval system which applies to the current branch program of the program tree. This system is further described below.

Another area of particular interest for the operator is the alarm display. Each active alarm is displayed on a single line on the alarm screen. More information about the failing equipment or system can be obtained from a program available on the touch button page for machine operation. The alarm system is fully described in [10].

14.1 The HELP facility for branch programs.

The help facility is activated by pressing the system button labelled HELP. It runs a NODAL program at the interactive level which hunts for the appropriate help text. The search starts with the program on the top of the stack and it is carried through all the way to the bottom i.e. to program TRUNK. The search stops as soon as it finds a valid help text. The text is then displayed on the large black and white screen in the right hand bay of the console. Valid help texts may be kept either in the string array TX of the branch program itself, or in one of the special indices on the library. These help files have the same names as the branch program. They contain the string array TX. It is the application programmers responsibility to set up the help files for his application and to maintain them. A service program to aid in the maintenance of the help files is provided in the program (LIB)<280>EDITOR which runs on the consoles.

15 Accelerator timing facilities.

Some console programs require synchronization of the program execution with the timing system of the accelerator. A timing event module is inserted in the CAMAC crate of each console. It creates a sequence of interrupts in the console computer as the accelerator cycle proceeds. NODAL provides a set of execution control commands related to the accelerator events.

In an interactive NODAL program the program execution can be synchronized with the accelerator cycle by means of the WAIT-CYCLE command which is described in [1]. The WAIT-CYCLE command takes two arguments: the accelerator event number and the delay between the event and the time the execution of the program shall resume.
It is often desirable to run a complete program at a certain instant in the accelerator cycle. RTRUN is one of the functions which allows this. The statement:

\[ \text{>RTRUN(P.NAME,2,3[1],0.4)} \]

will put the program P.NAME into the event queue. The program will run 0.4 seconds after event number 3 on cycle 1 has occurred in the accelerator cycle sequence. The second parameter is the default device number for the output from P.NAME. Another function REPEAT will ensure that the program is run at the specified event in every accelerator cycle which follows. Both these functions are described in [1]. The system of the consoles provides a separate working area for this kind of programs.

16 The local file BEEF.

A suite of interactive programs often needs a means of intercommunication when run in the consoles. A unique file named BEEF has been provided in the NODAL system of the console to satisfy this requirement. The storage medium is the primary memory of the computer itself. As full handshake is provided when a file is saved the file can be read immediately after being saved without any special precautions. The file size is limited to 4000 bytes. Data is saved by the usual NODAL commands. For example the statement:

\[ \text{>SAVE BEEF A B} \]

will save the data elements A and B. Likewise:

\[ \text{>LOAD BEEF} \]

will load whatever was previously stored in BEEF into the working area. The SAVE, LOAD, LDEF or OLD may of course be executed in different programs.

The following program shows an example of how BEEF can be used to capture a large amount of printable output which would otherwise have disappeared off the display screen. Once stored in BEEF the information is retrieved, processed and displayed under operator control.

1.02 TYPE "PROGRAM TO BREAK THE LIST OF FUNCTIONS UP INTO PAGES";
1.10 $ASEK "COMPUTER NAME" C; $SET LEGEND(0)="WAIT FOR AQUISITION"
1.30 SET FI=OPEN("W","BEEF"); SET ODEV=FI; IMEX($C) LISR
1.40 CLOSE(FI)
1.50 $SET LEGEND(0)=
2.10 SET F=open("R","BEEF")
2.30 SET ODEV=10; FOR J=1,100; DO 30

9.90 END

30.10 TYPE \12 "RESIDENT LIST IN COMPUTER " C", PAGE NUMBER:" %3 J !
30.20 FOR I=1,23; $SET S=INPUT(F):30.9; TYPE S !
30.40 $SET LEGEND(11)="\NEXT PAGE"
30.50 SET Z=BUTTON; $SET LEGEND(0)=; RETURN
30.90 END

17 Restrictions on programs.

The consoles serve the operators in an interactive environment and the restrictions placed on the programs may be different from what they are elsewhere in the control system. The console computer system is designed to interact directly with a human operator of the console.

Because of limited resources in the computer system there are limitations on the amount of memory a program can occupy and on the length of time a program can reserve buffer space. The latter is surveyed by a watch dog timer which aborts the offending program and creates an error when a time-out is detected.

A distinction is made between four classes of user programs in the consoles:

1. Interactive programs.
2. Scheduled programs.
3. Executed programs (EXECUTE from other computers).
4. Short executed programs (IMEX from other computers).

Normally, application programs are of the first and second class. The present basic restrictions on the four classes are:

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working area (words)</td>
<td>6000</td>
<td>4200</td>
<td>2500</td>
<td>2000</td>
</tr>
<tr>
<td>Defined function area (words)</td>
<td>6000</td>
<td>2500</td>
<td>2500</td>
<td>0</td>
</tr>
<tr>
<td>Time-out limits (sec.)</td>
<td>none</td>
<td>60</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Applications programs acquire data and issue control commands by remote execution. The part of the program which is executed in remote computers is subject to more severe restrictions than those listed
above. First, the part of the program which is executed remotely must be completed within three seconds. Second, while remote execution is taking place the console computer will allow only four seconds before it times out waiting for an acknowledgement from the remote computer. If that happens the computer link software will override the dialog and cause the failure return "REMITTED DATA LOST". Because of this constraint, a minimum of data processing should be done in the remote computer.

Another limitation is the amount of program and data which is allowed to be submitted to a remote computer. This limit is 900 words in case of the standard general purpose computers (GP's). The amount of data which can be remitted from a single execute is limited by the size of the working area (40000 words) in the remote computer (provided that the WAIT command follows promptly, see the paragraph below).

A less obvious restriction is the limited amount of buffer memory which is available for the computer link handling. All program and data transferred over the computer link pass through this buffer memory. There are 39 blocks of 64 words available for this purpose in the console computers i.e. 2560 words in total. The function FRBUC returns the number of free blocks available. The limitation in the number of buffer blocks can be avoided if the program follows each EXECUTE command by a companion WAIT command for the remitted data:

>EXECUTE(GP3) 10 A B; WAIT(GP3)

However, when the remote parts of the program are executed in several computers in "parallel" then one must ensure that there is room for all the remitted data in the buffer. The console allows for 12 such parallel executes at a time, but there is always a chance that the outstanding remitted data will absorb all the 39 buffer blocks.
References:


Appendix I. Software functions.

USER    Log a user into the control system.
SET USER=<CERN identity number>
Makes a user known to the control system.
The function will ask for a password to be typed in
from the keyboard.
Then the function looks up the user's section, capability
and default file index from the password file.
These entries are then written into the RT-description
of the interactive NODAL.
SET USER=0 logs the user out.

SECTN   Read/Write section number.
SET V=SECTN
SET SECTN=V
When SECTN is used in write mode then it will ask for
the section's password to be typed in on the keyboard.

CAPABI  Read the capability.
SET V=CAPABI
Reads the current user's capability word.

ENTERC  Log a user in from his badge.
CALL ENTERC
Reads the user's identity badge from the badge reader
and makes him known to the control system.
The function looks up the user's section, capability
and default file index from the password file.
These entries are then written into the RT-description
of the interactive NODAL.
The previous user is logged out if no identity badge
is present in the reader.
SCREEN
Define the view port for array oriented graphics.
CALL SCREEN(XU1,XU2,YU1,YU2)
The area limits must be given in screen graphics units:
0 <= XU <= 767
0 <= YU <= 575
The default setting covers the whole screen i.e.
XU1 = 0, XU2 = 767, YU1 = 0, YU2 = 575
The following restrictions apply:
IF XU2 <= XU1 THEN XU2 := XU1+1
IF YU2 <= YU1 THEN YU2 := YU1+1

WINDOW
Define the range of program data to see through the viewport.
CALL WINDOW(X1,X2,Y1,Y2)
The parameters must be in program data units.
The default setting gives a one to one correspondence
between the viewport and window i.e.
X1 = 0, X2 = 767, Y1 = 0, Y2 = 575
The following restrictions apply:
IF X1 > X2 THEN X2 := X1
IF Y1 > Y2 THEN Y2 := Y1

XSCAL
Read the horizontal scaling factor.
SET S=XSCAL

YSCAL
Read the vertical scaling factor.
SET S=YSCAL

XOFF
Read the horizontal offset.
SET O=XOFF

YOFF
Read the vertical offset.
SET O=YOFF

VALID
Check the parameters of 'SCREEN' and 'WINDOW'.
SET A=VALID
VALID=0 IF OK
VALID=1 IF BAD
DIAGRM  Plot a diagram from array data.
      CALL DIAGRAM(<array>,<basic display function>)
      Plot a diagram where the array index is the running coordinate
      along the horizontal axis. The value of each point is loaded
      from the corresponding array value. The position of the plot
      and its scaling is controlled by the last calls of SCREEN and
      WINDOW. The appearance of the plot is controlled by the second
      parameter which is the name of the predominant basic display
      function used by DIAGRM.
      The following basic functions are allowed:
      1. SOLID
      2. BOX
      3. POINT
      4. VERTIC
      5. HORIZO

      DIAGRM plots each basic shape within the interval;
      i-0.5 =< ... < i+0.5
      where i is the running index of the array.

YDOT  Plot the data in an array as a scatter diagram.
      CALL YDOT(A)
      Plots a scatter diagram where the horizontal components
      are given by the elements in the array 'A' and the
      vertical components are the corresponding array indices.
      Each point is effectively drawn by the elementary graphics
      function POINT and the form of the display can be
      programmed accordingly.

YPLOT  Plot the data in an array as a continuous curve.
       CALL YPLOT(A)
      Plots a continuous curve where the horizontal components
      are given by the elements in the array 'A' and the
      vertical components are the corresponding array indices.
      The points are effectively joined by straight line segments
      drawn by the elementary graphics function VECT.
DOT
Plot the data in an array as a scatter diagram.
CALL DOT(A)
Plot a scatter diagram where the vertical components are given by the elements in the array 'A' and the horizontal component are the corresponding array indices. Each point is effectively drawn by the elementary graphics function POINT and the form of the display can be programmed accordingly.

PLOT
Plot the data in an array as a continuous curve.
CALL PLOT(A)
Plots a continuous curve where the vertical components are given by the elements in the array 'A' and the horizontal components are the corresponding array indices. The points are effectively joined by straight line segments drawn by the elementary graphics function VECT.

XYDOT
Plot the data in two arrays as a scatter diagram.
CALL XYDOT(A,B)
Plot a scatter diagram where the horizontal components are given by the elements in the array 'A' and the horizontal components are given by the corresponding elements in the array 'B'.

XYPLOT
Plot the data of two arrays as a continuous curve.
CALL XYPLOT(A,B)
Plot a continuous curve where the horizontal components are given by the elements in the array 'A' and the vertical components are given by the corresponding elements in the array 'B'.

YHISTO
Plot a histogram.
CALL YHISTO(A)
Plot a histogram where the heights of the bins are given by the elements in the array 'A' and the corresponding array indices are the sample numbers. The bins are drawn horizontally and the samples run vertically.
HISTO  Plot a histogram.
CALL HISTO(A)
Plot a histogram where the height of the bins are given by the elements in the array 'A' and the corresponding array indices are the sample numbers. The bins are drawn vertically and the sample numbers run horizontally.

XAXIS  Draw a horizontal axis.
CALL XAXIS
The position of the axis and the tick marks are controlled by the last calls of 'SCREEN' and 'WINDOW'. The tick marks are annotated accordingly.

YAXIS  Draw a vertical axis.
CALL YAXIS
The position of the axis and the tick marks are controlled by the last calls of 'SCREEN' and 'WINDOW'. The tick marks are annotated accordingly.

ODEV  Set NODAL output stream
SET ODEV= the file number of the desired output file
An extended version of 'ODEV' initializes the display control parameters in the NODAL global area.

CLEAR  Clear the screen.
CALL CLEAR
Erases the screen and sets origin and the current position to bottom left hand corner of the screen.

ORIGIN  Set a new origin.
CALL ORIGIN(X,Y)
The default origin is in the bottom left hand corner of the screen.

MOVE  Move the current position.
CALL MOVE(X,Y)
The current position is moved to a new position.
POINT

Bright up a point.
CALL POINT(X,Y)
A small display form is drawn at the position [X,Y].
The default form is a single pixel but it can be changed.
See the description of the display control characters for programming details.
The current position is moved to [X,Y].

VECT

Draw a straight line.
CALL VECT(X,Y)
A straight line segment is drawn between the current position and the position [X,Y].
The current position is updated to [X,Y].

DOTTED

Draw a broken line.
CALL DOTTED(X,Y)
A broken line is drawn between the current position and [X,Y]. The number of pixels which are lit and unlit in the line is controlled by DOTLEN. The default is eight for both. The current position is set to [X,Y].

DOTLEN

Set dotlengths for 'DOTTED'.
CALL DOTLEN(ON,OFF)
The default settings are: ON=8 and OFF=8.

HORIZO

Draw a horizontal line.
CALL HORIZO(X)
Draws a straight horizontal line segment from the current position to a new position with the same vertical coordinate but with the horizontal coordinate X.
The current position is updated accordingly.

VERTIC

Draw a vertical line.
CALL VERTIC(Y)
Draws a straight vertical line segment from the current position to a new position with the same horizontal coordinate but with the vertical coordinate Y.
The current position is updated accordingly.
BOX  Draw a box.
CALL BOX(X,Y)
Draws a box where the straight line segment from the current position to [X,Y] is a diagonal.
The current position is set to [X,Y].

SOLID Draw a solid.
CALL SOLID(X,Y)
Draws a solid where the straight line segment from the current position to [X,Y] is a diagonal.
The current position is set to [X,Y].

CIRCLE Draw a circle.
CALL CIRCLE(R)
The circle is centered at the current position.

OCTANT Draw an octant.
CALL OCTANT(R,N)  N=1,...,8
The octant lies on a circle centered at the current position.

RMOVE Relative move.
CALL RMOVE(DX,DY)
Moves the current position relatively.

RVECT Draw a vector relativ to the current position.
CALL RVECT(DX,DY)

CHSIZE Set character size for software characters.
CALL CHSIZE(SZ)
SZ<0 rotates the orientation of the string 90 degrees anticlockwise.

TEXT NODAL interface to software generated characters.
CALL TEXT(CONCATENATION)
TVCNCT  Connect a TV-source to a screen.
   CALL TVCNCT(<source number>,<global screen number>)
   Source numbers and global screen numbers are
   according to the drawing CERN-SPS 5.2150.01.001.0.
   Display computer function.

APPEAR  Make a TV-source appear on a console screen.
   CALL APPEAR(<source number>,<console screen number>)
   The source numbers are according to the
   drawing CERN-SPS 5.2150.01.001.0.
   Possible console screen numbers are:
   15 to 18 (small black and white screens).
   Display computer function.

PCOPY   Copy the picture from one screen to another.
   CALL PCOPY(A,B)
   A = global number of source screen.
   B = global number of destination screen.
   The global screen numbers are according to
   the drawing CERN-SPS 5.2150.01.001.0.
   The source screen is not affected.
   Display computer function.

BREAK   Disconnect all picture sources from a console screen.
   CALL BREAK(<console screen number>)
   The valid console screen numbers are 15 to 18.
   Display computer function.

PURGE   Disconnect all picture sources from a screen.
   CALL PURGE(<global screen number>)
   The global screen numbers are according to
   the drawing CERN-SPS 5.2150.01.001.0.
   Display computer function.

WHERIS  Read the screen number connected to a display memory.
   SET SC=WHERIS(A)
   A = Device number of an allocatable DICO/DIME
   in the range 16 to 23.
   Display computer function.
KNOB
READ/WRITE THE VALUE OF THE KNOB
SET A=KNOB
SET KNOB=B
VALID RANGE IS:
30000 TO 30000 IN SPRING AND BRAKE MODE
938 TO 937 IN SWITCH MODE

MKNOB
SET MODE OF KNOB
CALL MKNOB(M,R)
THE VALID MODES ARE:
M = 1 CLICK SWITCH MODE
M = 2 SPRING RETURN MODE
M = 3 POTENTIOMETER MODE WITH BRAKE
THE PARAMETER 'R' IS A DUMMY IF M=1,2. IF M=3 THEN
R IS THE BRAKING FORCE IN THE RANGE 0 TO 255.

SKNOB
SET ENDSTOP VALUES FOR KNOB
CALL SKNOB(LOW,HIGH)

CFIND
FIND LINE/COLUMN NEAREST TO THE CURSOR
SET A=CFIND(ARR)
A = RETURNED INDEX TO THE ARRAY 'ARR'.
ARR = TWO DIMENSIONAL ARRAY CONTAINING
LINE COORDINATE ARR(J,1) AND COLUMN COORDINATE ARR(J,2)
OF A SELECTION OF CHARACTER POSITIONS.

LABEL
Write a text on a self-scan label.
$SET LABEL(I)="STRING"
The string is displayed on the self-scan label number I.
The label associated with the knob is number one.
Maximum length of text is 32 characters.
LEGEND
WRITE TOUCH BUTTON LEGEND
$SE LEGEND(N)=CONCATENATION
WRITE TOUCH BUTTON TEXT UNDER BUTTON N, (1-16)
N=0 CLEARS THE SCREEN AND WRITES THE CONCATENATION
AS TITLE TO THE PAGE (LINE 1). ALL BUTTONS ARE
DISABLED.
N=0 WRITES THE TEXT STRING UNDER THE BUTTON AND
ENABLES IT.
THE TEXT COVERS A MAXIMUM OF THREE LINES OF TEN
CHARACTERS AND EACH LINE IS CENTRED AUTOMATICALLY
THE SPECIAL CHACTER '\"' FORCES NEWLINE.
THE NORMAL FORMAT IS BLACK CHARACTERS ON A WHITE
BACKGROUND. THE CONTROL FUNCTION INVERT IS AVAILABLE.
IF THE STRING IS EMPTY, THE BUTTON IS CLEARED
AND DISABLED.

BUTS
READ BUTTON
SET Z=BUTS(G)
IMMEDIATE READ OF BUTTONS IN A GROUP.
Z ON RETURN HOLDS BUTTON NUMBER WITHIN A GROUP.
Z=0 MEANS THAT NO BUTTON WITHIN THE GROUP HAS
BEEN PRESSED. THE PARAMETER G INDICATES WHICH BUTTON
GROUP TO READ. G=1 (MAIN TOUCH BUTTONS) IS THE DEFAULT
WHEN THE NO PARAMETER IS SUPPLIED WITH THE CALL.
G=1 MAIN TOUCH-BUTTONS (default)
G=2 SPARE BUTTON GROUP
G=3 BALL BUTTON
G=4 KNOB BUTTON
G=5 SCREEN IDENTIFY BUTTONS

BUTTON
Wait for button to be pressed.
SET Z=BUTTON(G)
The program waits until a button in the group has been pressed.
Then BUTTON returns with the button number of the button
which was pressed.
See BUTS for a list of valid button groups.

COLOUR
RETURN LOGICAL NUMBER OF MAIN COLOUR SCREEN.
SE X=COLOUR
DAYTIM  Read date and time.
SET S=DAYTIM
Returns an 18 character string with the current date and time.
Date is nine characters long and time is eight characters long
i.e. 12 JAN 77 11:12:54

PINXIT  Draw an encoded character picture.
CALL PINXIT(A)
Dimension of A is: DIMENS-INTEGER A(64,24)
The array elements contain the 16 bit hardware command words
of the Chr. Roving format.
The data in the array is best entered by means of the interactive
program (LIB)<96>BUILD.

LINE  String function to set line position of display.
SET S=LINE(Y)
This is equivalent to SET S=\11\Y

COLUMN String function to set column position of display.
SET S=COLUMN(X)
This is equivalent to SET S=\9\X

BHOLD  Read the number of characters in device buffer.
SET N=BHOLD(<logical number>)
If no parameter is supplied then IDEV is assumed.

FRBUC  Free buffer counter.
SET A=FRBUC
Returns the number of free buffers available
for the computer link handling.

CURC  Read/write horizontal cursor position in column units.
SET C=CURC
SET CURC=C
The column numbering runs from left to right 1 to 64.
The conversion formulae between column units
and graphics screen units are:
C := INT(X/12)+1
X := 12*C-6
CURL  Read/write cursor vertical cursor position in line units.
     SET L=CURL
     SET CURL=L
     The line numbering runs from top to bottom 1 to 24.
     The conversion formulae between line units
     and graphics screen units are:
     \[ Y := (24-L) \times 24 + 12 \]
     \[ L := 24 - \text{INT}(Y/24) \]

HCURS  READ OR WRITE THE HORIZONTAL CURSOR POSITION.
     SET X=HCURS
     SET HCURS=X
     HCURS WORKS IN GRAPHICS UNITS 0 TO 767.

VCURS  READ OR WRITE THE VERTICAL CURSOR POSITION.
     SET Y=VCURS
     SET VCURS=Y
     VCURS WORKS IN GRAPHICS UNITS 0 TO 575.

HBALL  READ OR WRITE HORIZONTAL BALL POSITION.
     SET HBALL=A
     SET B=HBALL

VBALL  READ OR WRITE VERTICAL BALL POSITION.
     SET VBALL=C
     SET D=VBALL
MCURS  Set the operational mode of the cursor.
       CALL MCURS(M,H)
       The valid display modes (M) are:
       M=0,1 - suppress cursor display
       M=2  - horizontal line
       M=4  - vertical line
       M=6  - cross, horizontal and vertical line
       M=8  - solid
       M=1 - as mode M but blinking

       The second parameter (H) hooks the cursor to
       the movements of the tracker ball.
       The valid values for hooking are:
       H=0  unhook the cursor from the ball
       H=1  hook the cursor to horizontal ball movements
       H=2  hook the cursor to vertical ball movements
       H=3  hook the cursor to both ball movements

       After the call of MCURS the cursor size is set to default
       values. The lines have a width of one. The solid (M=8)
       has a height of 20 and a width of 12.

CURSIZ  SET CURSOR WIDTH AND HEIGHT
       CALL CURSIZ(W,H)
       WIDTH CAN BE SET FROM 0 TO 767
       HEIGHT CAN BE SET FROM 0 TO 575

CURCON  CURSOR SWITCH FUNCTION
       CURCON(0) - DISCONNECT THE CURSOR FROM ALL SCREENS
       CURCON(10) - CONNECT THE CURSOR TO THE COLOUR SCREEN
       CURCON(14) - CONNECT THE CURSOR TO THE LARGE B/W-SCREEN

CAMAC  Read/write of a CAMAC module.
       SET CAMAC(C,N,A,F)=Z
       SET Z=CAMAC(C,N,A,F)
       CAMAC status register is returned for control functions.
       The parameters are:
       C = CAMAC crate
       N = slot
       A = sub address
       F = function
       Warning: Use of the CAMAC function may upset the console hardware.
TRUNK
Clears the program stack.
CALL TRUNK
The current program will then be set to the default trunk of the program tree.

NEXT
The next program is pushed onto the stack.
CALL NEXT(filename)
The filename of the next program is pushed onto the stack and run from section number one. THE INDEX IS CHANGED.

THISP
Return the filename of the current program.
$SET S=THISP
Returns a filename from the program stack.
An optional parameter allows reading of the names at any depth in the stack; zero reads top level, one reads next down and so on. If no parameter is supplied then top element is read.

BACK
The previous program is run.
CALL BACK
The current program name is chopped off the stack and the previous program is run from section number two.

BIT
BIT FUNCTION
SET BIT(N,A)=Z; SET Z=BIT(N,A)
A=DESTINATION, CAN BE ANY SETTABLE NAMED ITEM
N=BIT NUMBER, 0 TO 15
SET OR READ A SPECIFIC BIT IN A SUPPOSED 16 BIT WORD

IOR
DO LOGICAL INCLUSIVE OR
SET Z=IOR(M,N)
INCLUSIVE OR OF (M,N) INTO Z
RETURNS INCLUSIVE 'OR' OF TWO INTEGER ARGUMENTS

AND
DO LOGICAL AND
SET Z=AND(M,N)
LOGICAL PRODUCT OF (M,N) INTO Z
RETURNS AND OF TWO INTEGER ARGUMENTS
NEG      DO LOGICAL COMPLEMENT
SET Z=NEG(M)
INVERTS ALL BITS; RETURNS COMPLEMENT OF INTEGER ARGUMENT

SHIFT    DO LOGICAL SHIFT
SET Z=SHIFT(EXPR,N)
EXPR IS ANY EXPRESSION EVALUATED AND ROUNDED TO INTEGER
N IS NUMBER OF SHIFTS, +VE FOR SHIFT LEFT, -VE FOR SHIFT RIGHT
LOGICAL SHIFT, IE. ZEROS PUT INTO VACATED BITS

ARSIZE   RETURN SIZE OF ARRAY
GIVES NO OF ELEMENTS IN ARRAY
ONE PARAMETER, NAME OF ARRAY

ERROR    MANIPULATE ERROR SITUATION
SET ERROR=EXPRESSION, PROVOKES NODAL ERROR
POSITIVE ERROR NUMBER MUST BE GIVEN
OTHERWISE ERR37 VALUE OUT OF RANGE IS GIVEN
SET Z=ERROR, RETURNS ERROR SET ON DO ERROR JUMP

ERRLIN   RETURN ERROR LINE AS NUMBER
SET LN=ERRLIN
RETURNS ZERO IF NO ERROR SINCE LAST DO COMMAND

LOC      SET OR READ MEMORY LOCATION
SET LOC(I)=[XXXXXX]
WARNING! THIS FUNCTION CAN DAMAGE THE CONTROL SYSTEM
********** IT IS FOR THE USE OF SPECIALISTS ONLY

ERMS     RETURN ERROR MESSAGE
ERMS(N) WHERE N = ERROR NUMBER

STRARG   STRING ARGUMENT IN GLOBAL AREA
USES 21 DECIMAL, 25 OCTAL GLOBALS FROM 9STAG
9STAG CONTAINS WRITER POINTER
9STAG+1 TO 9STAG+24 CONTAIN STRING, SPACE FOR 40 CHARACTERS
READ-WRITE STRING FUNCTION
NODLIN GET OR CREATE NODAL LINES
$SET NODLIN(LN)=STRING, CREATES LINE OF NUMBER LN
$SET S=NODLIN(LN), RETURNS LINE WITHOUT NUMBER
READ-WRITE STRING FUNCTION
STRING FUNCTION FAILURE IF NO SUCH LINE BUT THERE IS ONE GREATER

CHAIN RUN CHAINED FILES
CALL CHAIN FN [FN] [FN].....
RUNS CONSECUTIVE FILES RETAINING DATA IN TEXT BUFFER
ANY NUMBER OF FILES MAY BE RUN - UPTO END OF COMMAND.

COPY COPY ARRAY
COPY(A,B,SA,SB)
COPY ARRAY 'A' INTO ARRAY 'B'.
START WITH ELEMENT NUMBER 'SA' IN 'A', AND WITH ELEMENT
NUMBER 'SB' IN 'B'.
THE COPY IS TERMINATED WHEN EITHER ARRAY IS EXHAUSTED.

MAX FIND MAXIMUM VALUE OF AN ARRAY
SET A=MAX(ARR)
ARR CAN BE INTEGER OR REAL ARRAY.
IT IS ASSUMED TO BE ONE DIMENSIONAL.

MIN FIND MINIMUM VALUE OF AN ARRAY
SET A=MIN(ARR)
ARR CAN BE INTEGER OR REAL ARRAY.
IT IS ASSUMED TO BE ONE DIMENSIONAL.

FIRST LOOK FOR CONDITIONED ELEMENT
SET Z-FIRST(A [BOOLEAN OPERATOR] [EXPRESSION])
A = ARGUMENT ARRAY
EQUIVALENT NODAL PROGRAM:
SET I=0; SET Z=N
WHILE I<N; SE I=I+1; IF A(I)[BOOL.OP.][EXPR.]; SE Z=I; SE I=N
GETV
GET VARIABLES FROM FILE
GETV(S1,S2)  TWO STRING ARGUMENTS
GETV('FILENAME', 'ARRAY1 ARRAY2 V1 V2')
LOADS FROM FILENAME ONLY VARIABLES IN LIST
NO ERROR IF NAME NOT FOUND

LISV
LIST PROGRAM VARIABLES
FOLLOWING FORMAT FOR OUTPUT
COL 1  VARIABLE NAME
COL 2  SIZE OF NODAL ELEMENT
COL 3  TYPE
COL 4  CURRENT VALUE OF VARIABLE

LISD
LIST DEFINED FUNCTIONS
ALSO GIVES SIZE OF DEFINED FUNCTION AREA

LISR
LIST USER FUNCTIONS
FOLLOWING FORMAT FOR OUTPUT
COL 1  CORE ADDRESS (OCTAL) OF FUNCTION
COL 2  NAME OF FUNCTION
COL 3  SIZE (DECIMAL) OF FUNCTION
COL 4  FUNCTION TYPE AND CALLING SEQUENCE
LIST TERMINATES WITH OCTAL ADDRESS OF END OF ELEMENT LIST,
THE TOTAL NUMBER OF FUNCTIONS, AND THE TOTAL NUMBER
OF WORDS USED.

LISA
LIST INBUILT FUNCTIONS
FORMAT SAME AS LISR

HELP
LIST NODAL COMMANDS AVAILABLE
GIVES NODAL COMMANDS AND THEIR MINIMUM ABBREVIATION.
FOUR COMMANDS PER LINE.

LISC
LIST COMPUTER NAMES AND NUMBERS
GIVES THE STANDARD MNEUMONICS FOR THE COMPUTERS IN THE
SYSTEM AND THEIR CORRESPONDING NUMBERS.
ALPHA  PRE-DEFINED STRING VARIABLE  
        RETURNS A TO Z

NUM    PRE-DEFINED STRING STRING VARIABLE  
        RETURNS 0 TO 9

ASCII  GET ASCII VALUE  
        VALUE=ASCII(CONC)  
        FORMS ASCII VALUE OF STRING BY ADDITION OF ALL CHARACTERS

SIZE   FIND SIZE OF CONCATENATION  
        VALUE=SIZE(CONC)  
        RETURNS NUMBER OF CHARACTERS IN A CONCATENATION

SUBS   SUBSTITUTE STRING  
        STRING = SUBS(BEGINNING, END, CONC)  
        OR $SET SUBS(BEGINNING, END, STRING) = STRING  
        RETURN/SET SPECIFIED SUBSTRING OF GIVEN CONCATENATION

EVAL   EVALUATE STRING EXPRESSION  
        VALUE=EVAL(CONC)  
        TREATS CONC AS AN ARITHMETIC EXPRESSION AND  
        RETURNS ITS VALUE

FIND   FIND INDEX  
        VALUE=FIND(ARRAY-NAME, CONC)  
        RETURNS INDEX OF CONC IN ARRAY  
        1 IF NOT FOUND

FINDS  FIND INDEX, GIVEN A UNIQUE SUBSTRING  
        VALUE=FINDS(ARRAY-NAME, CONC)  
        RETURNS INDEX OF CONC IN ARRAY  
        CONC MAY BE A SUBSTRING OF THE ARRAY ELEMENT  
        RETURNS -2 IF THE SUBSTRING IS NOT UNIQUE  
        RETURNS -1 IF NOT FOUND
SORT

SORT STRING ARRAY
NULL STRING = SORT(ARRAY-NAME, + OR -)
SORT STRING IN ASCENDING OR DESCENDING ORDER

RUNOFF

TEXT PROCESSOR FUNCTION.
$SET S=RUNOFF(CONCATENATION)
THE ALPHABETICS IN THE CONCATENATION ARE CONVERTED
TO LOWER CASE. HOWEVER, THE CONVERSION OBEYS CERTAIN
SPECIAL CONTROL CHARACTERS:
/ (SLASH) THE NEXT CHARACTER IS CONVERTED
TO UPPER CASE.
// (TWO SLASHES) THE SUCCEEDING CHARACTERS ARE ALL
CONVERTED TO UPPER CASE.
\ (TWO BACK SLASHES) THE SUCCEEDING CHARACTERS ARE ALL
CONVERTED TO LOWER CASE.
# (HASH) ONE SPACE.
_ (BACK ARROW) INTERPRET THE NEXT CHARACTER
LITERALLY.
ALL OTHER CHARACTERS ARE COPIED UNMODIFIED EXCEPT
THAT PARITY IS REMOVED.

ARB

MATCH ANY CHAR OR CHAR STRING

POS

MATCH A SPECIFIC LOCATION IN STRING
POS(I)
I STARTS WITH 0 AT BEGINNING OF STRING

RPOS

REVERSE POSITION MATCH
RPOS(I)
SIMILAR TO POS(I), BUT POSITION I IS COUNTED FROM
END OF STRING.
RPOS(0) IS JUST TO THE RIGHT OF THE LAST CHARACTER

LEN

LENGTH MATCH
LEN(I)
MATCHES ANY STRING OF LENGTH (I)
TAB
TAB(I)
MATCH UP TO POSITION I

RTAB
REVERSE TAB
RTAB(I)
SIMILAR TO TAB, BUT MATCHES UP TO THE
I-TH POSITION FROM THE END OF THE STRING

SPAN
SPAN(CONC)
MATCH THE LONGEST RUN OF CHARACTERS CONTAINED IN
CONC, STARTING FROM THE CURRENT CURSOR POSITION

BREAK
BREAK(CONC)
MATCH UPTO BUT NOT INCLUDING ANY BREAK CHARACTER
CONTAINED IN CONC

ANY
ANY(CONC)
MATCH ANY SINGLE CHAR CONTAINED IN CONC

NOTANY
NOTANY(CONC)
MATCH ANY SINGLE CHARACTER NOT CONTAINED IN CONC

ABORT
ABORT PATTERN
IF TRIED CAUSES PATTERN MISMATCH AND SO CAUSES OTHER
ALTERNATIVES TO BE TRIED

FAIL
FAIL PATTERN
IF TRIED CAUSES THE COMPLETE FAILURE OF THE $MATCH
COMMAND I.E. NO OTHER ALTERNATIVES ARE TRIED
INBT  NODAL FUNCTION TO INPUT A BYTE FROM A FILE
VALUE=INBT(F)  WHERE F = FILE LOGICAL NUMBER
RETURNS FULL 8 BIT VALUE OF BYTE AS POSITIVE INTEGER

OUTBT  OUTPUT A BYTE TO A FILE
SET OUTBT(F)=Z
EXPRESSION Z IS FIXED TO INTEGER AND
LOWER 8 BITS OUTPUT TO FILE
F = FILE LOGICAL NUMBER

OPEN  VALUE = OPEN('R' OR 'W' OR 'A',STRING)
OPEN A FILE GIVEN BY THE STRING
FOR READ, WRITE OR APPEND
RETURN FILE NUMBER

CLOSE  Close a file or all opened files
CLOSE(F), closes file F
CLOSE(-1), closes all files

OUTPUT OUTPUT(FILE NUMBER)=STRING
OUTPUT A STRING FOLLOWED BY CRLF

OUTC  OUTC(FILE NUMBER)=STRING
OUTPUT A STRING TO A FILE
$SET OUTC(F)="HELLO"

INPUT  STRING = INPUT [(FILE NUMBER)]
INPUT A LINE FROM A FILE
$SET S=INPUT(F)

INPC  STRING = INPC [(FILE NUMBER)]
INPUT A CHARACTER FROM A FILE
$SET S=S INPC(F)
RWDATA: READ OR WRITE FORTRAN FORMAT DATA
      CALLED AS RWDATA(CONC,FILE,DEV,LIST)
      CONC IS RFIRLWFWLIFAL FOR READ, WRITE, OR APPEND
      ON FILE (FILE OPENED FIRST, CLOSED AFTERWARDS),
      OR A LOGICAL NUMBER.
      SECOND PARAMETER IS CONC FOR FILENAME,
      OR EXPRESSION FOR LOGICAL NUMBER.
      LAST PARAMETER IS A LIST OF VARIABLES OR ARRAYS

RTRUN: RUN A FILE ONCE AT A GIVEN EVENT.
       RTRUN(Filename,ODEV,EVENT,DELAY)
       EVENT CAN BE OF THE FORM: EVENT[CYCLE]

REPEAT: RUN A FILE AT EVERY OCCURRENCE OF AN EVENT.
        REPEAT(Filename,ODEV,EVENT,DELAY)
        EVENT CAN BE OF THE FORM: EVENT[CYCLE]

CTIME: READ THE TIME FROM START OF LAST CYCLE IN SECONDS.
       (TIME ELAPSED SINCE THE LAST EVENT 0)
       SET C=CTIME
       THE START OF THE CYCLE IS EVENT NUMBER 0.

CYLENG: READ THE LENGTH OF THE PREVIOUS CYCLE.
        (TIME ELAPSED BETWEEN THE TWO LAST EVENT 0)
        SET C=CYLENG
        THE TIME IS IN SECONDS.

LISP: LIST STATE OF NODAL RT-PROGRAMS

SCHEDL: SCHEDULE FILE BASED NODAL PROGRAM
       SCHEDL(Filename,ODEV,TIME,INTV)
       THE ODEV FOR A FIXED RT DESCRIPTION IS UNCHANGED.

KILL: ABORT SCHEDULED PROGRAM
       KILL(NAME)
       ABORTS CURRENT EXECUTION OF A PROGRAM
DISCON DISCONNECT NODAL PROGRAM.
CALL DISCON(NAME)
DISCONNECTS A PROGRAM FROM ANY EXECUTION QUEUE.

TIME READ-OUT OF TIME IN SECS
SET VALUE = TIME

SETATI SET INTERNAL CLOCK DOUBLE WORD
CALL SETATI(TIMX)

RTIME READ THE TIME FROM RESTART IN SECONDS.
SET C=RTIME
MACHST MACHINE STATUS INFORMATION ARRAY

M.W. TYRRELL.

SE MACHST(A,B)=Z; SE Z=MACHST(A,B)

WHERE:


THE VALUES OF THIS ARRAY FOLLOW THOSE OF THE ARRAY "MS"
CONTAINED IN THE MASTER CYCLE FILE, REF. SPS/AOP/NOTE/80-17
"DESCRIPTION OF THE MASTER CYCLE FILE & THE NEXT CYCLE FILE".

MACHST REPRESENTS THE MAXIMUM PHYSICAL STATE THAT
THE MACHINE CAN ATTAIN AT ANY POINT IN TIME.

EACH TIME THE MACHINE STATUS IS CHANGED IN THE MCR, THE PROGRAM
CONTROLLING THIS CHANGE UPDATES MACHST IN ALL COMPUTERS.

BIT DESCRIPTION OF THE ELEMENTS OF MACHST:

MACHST(1,1):

BIT NO. STATUS
0 NO BEAM
1 FIXED TARGET
2 COLLIDER
3 CRITICAL PERIOD

MACHST(2,1):

BIT NO. BIT 1 OF MACHST(1,1) SET BIT 2 OF MACHST(1,1) SET
0 NO EXTRACION PROTONS
1 SE1 PILOT P BAR
2 SE2 DENSE P BAR
3 SE3 INJ. COAST
4 SE4 LOW BETA 4
5 SE5 LOW BETA 5
6 FE1
7 FE2
8 FE3
9 FS
10 FFS
11 WEXTT TESTS
12 NEXTR TESTS

MACHST(3,1):

BIT NO. BIT 1 OF MACHST(1,1) SET BIT 2 OF MACHST(1,1) SET
0 WEST TED MULTIBUNCH
1 NORTH TED ONE BUNCH
2 T1 INJECT P
3 T2 EXTRACT P
4 T3 WEST TED
5 T4
6 T5
7 T6 INJECT PBAR
8 T7 STORE
9 T8 PHYSICS 4
10 T9 PHYSICS 5
11 T10
12 T11
13 TIM
NODAL TYPE 4 INTEGER ARRAY.

BEAMST BEAM STATUS INFORMATION ARRAY
M.W.TYRRELL.
SE BEAMST(A,B)=Z; SE Z=BEAMST(A,B)
WHERE:
BEAMST IS A 3*2 INTEGER ARRAY, "A" VALUES 1-3,
"B" VALUES 1-2.
THE BIT DEFINITIONS OF THIS ARRAY EQUAL THOSE OF THE ARRAY
MACHT. THE MAXIMUM PHYSICAL DESCRIPTION OF BEAMST CANNOT
EXCEED THAT OF MACHT. BEAMST REPRESENTS THE HIGHEST ALLOWED
PHYSICAL STATE OF THE BEAM AT ANY POINT IN TIME.
EACH TIME THE BEAM STATUS IS CHANGED IN THE MCR, THE PROGRAM
CONTROLLING THIS CHANGE UPDATES BEAMST IN ALL COMPUTERS.
NODAL TYPE 4 INTEGER ARRAY.

SYSTYP COMPUTER CONFIGURATION TYPE DEFINITION.
M.W.TYRRELL.
SE SYSTYP=Z; SE Z=SYSTYP
WHERE:
"Z" IS THE TYPE DEFINITION.
SYSTEM TYPES:
Z DESCRIPTION
0 GP
1 CONSOLE
2 SERVICE
3 TSS
NODAL TYPE 2 READ/WRITE VARIABLE.

WSTK WRITE STACK STATUS FOR ALL PARTITIONS
WSTK LISTS
THE PRIORITY LEVELS,
THE BEGINNING, THE END AND THE MAXIMUM USED SO FAR
OF THE WAITING STACKS,
THE ADDRESS (,THE NAME) AND THE STATE OF THE PROGRAMS.
IT ALSO REPORTS IF THERE IS ANY PROGRAM WAITING
IN THE EXECUTION QUEUE, BUT NOT ALL.

FROMC RETURN THE COMPUTER WHICH HAS SUBMITTED THE JOB
ERROR IF NOT A REMOTE JOB
LOCAL RETURN YOUR LOCAL COMPUTER NUMBER

INDEX READ/SET DEFAULT FILE INDEX NUMBER
ERROR IF OUTSIDE THE RANGE 0-511

OPENL OPEN THE LOCAL COMPUTER'S LINK TO THE MHC

CLOSNL CLOSE THE LOCAL COMPUTER'S LINK TO THE MHC

DLSTAT GIVE STATUS OF DATA LINKS
SET Z=DLSTAT(CN)
CN IS COMPUTER NUMBER, Z=1 IF LINK OPEN, Z=0 IF NOT.
THE LOCAL COMPUTER MUST BE OPEN TO THE NETWORK TO GET
TRUE STATUS OF OTHER LINKS.

DATE PRINT DATE
OUTPUTS DATE AS YYYY-MM-DD-HH:MM:SS

ADATE GET DATE AS INTEGER ARRAY
ADATE(ARRAY) RETURNS ARRAY(1)=YEAR
(2)=MONTH
(3)=DAY
(4)=HOUR
(5)=MINUTES
(6)=SECONDS

UKSYST UNLOCK SYSTEM TABLES AND FUNCTION LOC
UKSYST IS A PROTECTED FUNCTION, RESERVED TO SECMP AND 24.
IT TURNS TO READ/WRITE THE TYPE OF ARRAYS CABCX AND CNAMT
TABCI
AND OF FUNCTION LOC.

FCOPY COPY FILE OF ARBITRARY FORMAT
FCOPY("SOURCE","DESTINATION")
Appendix II. Character codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Note</th>
<th>Character or Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>*</td>
<td>Red foreground colour.</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>Green foreground colour.</td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td>Magenta foreground colour.</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>White foreground colour.</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Home the writing position (top left character).</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>Load the column position from the next character.</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Line feed.</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Load the line position from the next character.</td>
</tr>
</tbody>
</table>
| 12   |      | Form feed:  
|      |      | White foreground colour,  
|      |      | black background colour,  
|      |      | normal display mode (non-inverted),  
|      |      | no case shift,  
|      |      | normal character direction,  
|      |      | standard character set,  
|      |      | clear the screen,  
|      |      | home the writing position. |
| 13   |      | Carriage return. |
| 14   |      | Normal display mode (as opposed to inverted). |
| 15   |      | Inverted mode i.e. foreground and background colours are swapped. |
| 16   |      | Load character control from the next character.  
|      |      | Bits 0-1 contains writing direction for characters:  
|      |      | 0 = left to right,  
|      |      | 1 = bottom to top, |

*) These codes apply to colour displays only.
2 = right to left,
3 = top to bottom.
Bits 3-5 select the character set:
0 = extended ASCII character set,
1 = Greek/mathematical character set.
2 = high density extended ASCII character set
   for full resolution display,
3 = high density greek/mathematical character
   set for full resolution display.

19

Start graphics control block.
This code applies only to DICO graphics display.
This character is the first of a graphics
control block. The succeeding character is a
graphics control code. Then follows a number of
parameters determined by the graphics control
code. The list below shows the values of the
graphics control codes, the corresponding
display function name (if applicable) and the
parameter(s):

<table>
<thead>
<tr>
<th>Graphics control code</th>
<th>Display function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CLEAR</td>
</tr>
<tr>
<td>4</td>
<td>ORIGIN(X,Y)</td>
</tr>
<tr>
<td>5</td>
<td>MOVE(X,Y)</td>
</tr>
<tr>
<td>6</td>
<td>POINT(X,Y)</td>
</tr>
<tr>
<td>7</td>
<td>VECT(X,Y)</td>
</tr>
<tr>
<td>8</td>
<td>DOTTED(X,Y)</td>
</tr>
<tr>
<td>9</td>
<td>DOTLEN(ON,OFF)</td>
</tr>
<tr>
<td>10</td>
<td>HORIZO(X)</td>
</tr>
<tr>
<td>11</td>
<td>VERTIC(Y)</td>
</tr>
<tr>
<td>12</td>
<td>BOX(X,Y)</td>
</tr>
<tr>
<td>13</td>
<td>SOLID(X,Y)</td>
</tr>
<tr>
<td>14</td>
<td>CIRCLE(R)</td>
</tr>
<tr>
<td>15</td>
<td>OCTANT(R,N)</td>
</tr>
<tr>
<td>16</td>
<td>select point shape.</td>
</tr>
<tr>
<td></td>
<td>(see below)</td>
</tr>
<tr>
<td>20</td>
<td>RMOVE(dx,dy)</td>
</tr>
<tr>
<td>21</td>
<td>RVECT(dx,dy)</td>
</tr>
</tbody>
</table>

X = absolute horizontal coordinate
Y = absolute vertical coordinate
dX = increment of horizontal coordinate
dY = increment of vertical coordinate
ON = number of pixels to draw in foreground colour
OFF = number of pixels to draw in background colour

*) These codes apply to colour displays only.
R = radius
N = octant number 1 to 8 anticlockwise

Point shapes selectable from the byte following graphics control byte 16 are as follows:

0 = single point.
1 = big solid, 5 by 5.
2 = small solid, 3 by 3, in erased solid, 5 by 5.
3 = single point in erased solid, 5 by 5.
4 = erased solid, 5 by 5
5 = big diamond in erased solid, 5 by 5.
6 = small diamond in erased solid, 5 by 5.
7 = big box, 5 by 5, in erased solid, 5 by 5.
8 = small box, 3 by 3, in erased solid, 5 by 5.
9 = point up triangle in erased solid, 5 by 5.
10 = point down triangle in erased solid, 5 by 5.
11 = big circle in erased solid, 5 by 5.
12 = big filled circle in erased solid, 5 by 5.
13 = Greek cross in erased solid, 5 by 5.
14 = St. Andrea cross in erased solid, 5 by 5.
15 = right arrow in erased solid, 7 by 5.
16 = up arrow in erased solid, 5 by 7.
17 = left arrow in erased solid, 7 by 5.
18 = down arrow in erased solid, 5 by 7.
19 = left triangle in erased solid, 3 by 5.

These shapes are displayed by calling the POINT graphics function. The display of the shape is centered around the address of the point.

20 Save current display parameters.
21 Restore display parameters.
25 Increment column position.
26 Decrement column position.
28 Decrement line position.
29 Increment line position.

*) These codes apply to colour displays only.
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>No change of alphabetics to lower case.</td>
</tr>
<tr>
<td>31</td>
<td>Change alphabetics to lower case.</td>
</tr>
<tr>
<td>32</td>
<td>Printed characters according to chosen character set.</td>
</tr>
</tbody>
</table>
### Appendix III. Character fonts.

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>$</td>
</tr>
<tr>
<td>33</td>
<td>%</td>
</tr>
<tr>
<td>34</td>
<td>&amp;</td>
</tr>
<tr>
<td>35</td>
<td>'</td>
</tr>
<tr>
<td>36</td>
<td>(</td>
</tr>
<tr>
<td>37</td>
<td>)</td>
</tr>
<tr>
<td>38</td>
<td>*</td>
</tr>
<tr>
<td>39</td>
<td>+</td>
</tr>
<tr>
<td>40</td>
<td>,</td>
</tr>
<tr>
<td>41</td>
<td>-</td>
</tr>
<tr>
<td>42</td>
<td>.</td>
</tr>
<tr>
<td>43</td>
<td>/</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td>1</td>
</tr>
<tr>
<td>46</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>3</td>
</tr>
<tr>
<td>48</td>
<td>4</td>
</tr>
<tr>
<td>49</td>
<td>5</td>
</tr>
<tr>
<td>50</td>
<td>6</td>
</tr>
<tr>
<td>51</td>
<td>7</td>
</tr>
<tr>
<td>52</td>
<td>8</td>
</tr>
<tr>
<td>53</td>
<td>9</td>
</tr>
<tr>
<td>54</td>
<td>:</td>
</tr>
<tr>
<td>55</td>
<td>;</td>
</tr>
<tr>
<td>56</td>
<td>&lt;</td>
</tr>
<tr>
<td>57</td>
<td>=</td>
</tr>
<tr>
<td>58</td>
<td>&gt;</td>
</tr>
<tr>
<td>59</td>
<td>?</td>
</tr>
<tr>
<td>60</td>
<td>?</td>
</tr>
<tr>
<td>61</td>
<td>A</td>
</tr>
<tr>
<td>62</td>
<td>B</td>
</tr>
<tr>
<td>63</td>
<td>C</td>
</tr>
<tr>
<td>64</td>
<td>D</td>
</tr>
<tr>
<td>65</td>
<td>E</td>
</tr>
<tr>
<td>66</td>
<td>F</td>
</tr>
<tr>
<td>67</td>
<td>G</td>
</tr>
<tr>
<td>68</td>
<td>H</td>
</tr>
<tr>
<td>69</td>
<td>I</td>
</tr>
<tr>
<td>70</td>
<td>J</td>
</tr>
<tr>
<td>71</td>
<td>K</td>
</tr>
<tr>
<td>72</td>
<td>L</td>
</tr>
<tr>
<td>73</td>
<td>M</td>
</tr>
<tr>
<td>74</td>
<td>N</td>
</tr>
<tr>
<td>75</td>
<td>O</td>
</tr>
<tr>
<td>76</td>
<td>P</td>
</tr>
<tr>
<td>77</td>
<td>Q</td>
</tr>
<tr>
<td>78</td>
<td>R</td>
</tr>
<tr>
<td>79</td>
<td>S</td>
</tr>
<tr>
<td>80</td>
<td>T</td>
</tr>
<tr>
<td>81</td>
<td>U</td>
</tr>
<tr>
<td>82</td>
<td>V</td>
</tr>
<tr>
<td>83</td>
<td>W</td>
</tr>
<tr>
<td>84</td>
<td>X</td>
</tr>
<tr>
<td>85</td>
<td>Y</td>
</tr>
<tr>
<td>86</td>
<td>Z</td>
</tr>
<tr>
<td>87</td>
<td>[</td>
</tr>
<tr>
<td>88</td>
<td>]</td>
</tr>
<tr>
<td>89</td>
<td>^</td>
</tr>
<tr>
<td>90</td>
<td>_</td>
</tr>
<tr>
<td>91</td>
<td>`</td>
</tr>
<tr>
<td>92</td>
<td>a</td>
</tr>
<tr>
<td>93</td>
<td>b</td>
</tr>
<tr>
<td>94</td>
<td>c</td>
</tr>
<tr>
<td>95</td>
<td>d</td>
</tr>
<tr>
<td>96</td>
<td>e</td>
</tr>
<tr>
<td>97</td>
<td>f</td>
</tr>
<tr>
<td>98</td>
<td>g</td>
</tr>
<tr>
<td>99</td>
<td>h</td>
</tr>
<tr>
<td>100</td>
<td>i</td>
</tr>
<tr>
<td>101</td>
<td>j</td>
</tr>
<tr>
<td>102</td>
<td>k</td>
</tr>
<tr>
<td>103</td>
<td>l</td>
</tr>
<tr>
<td>104</td>
<td>m</td>
</tr>
<tr>
<td>105</td>
<td>n</td>
</tr>
<tr>
<td>106</td>
<td>o</td>
</tr>
<tr>
<td>107</td>
<td>p</td>
</tr>
<tr>
<td>108</td>
<td>q</td>
</tr>
<tr>
<td>109</td>
<td>r</td>
</tr>
<tr>
<td>110</td>
<td>s</td>
</tr>
<tr>
<td>111</td>
<td>t</td>
</tr>
<tr>
<td>112</td>
<td>u</td>
</tr>
<tr>
<td>113</td>
<td>v</td>
</tr>
<tr>
<td>114</td>
<td>w</td>
</tr>
<tr>
<td>115</td>
<td>x</td>
</tr>
<tr>
<td>116</td>
<td>y</td>
</tr>
<tr>
<td>117</td>
<td>z</td>
</tr>
<tr>
<td>118</td>
<td>{</td>
</tr>
<tr>
<td>119</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>}</td>
</tr>
<tr>
<td>121</td>
<td>^</td>
</tr>
<tr>
<td>122</td>
<td>_</td>
</tr>
<tr>
<td>123</td>
<td>`</td>
</tr>
<tr>
<td>124</td>
<td>~</td>
</tr>
<tr>
<td>125</td>
<td>@</td>
</tr>
<tr>
<td>126</td>
<td>#</td>
</tr>
<tr>
<td>127</td>
<td>$</td>
</tr>
<tr>
<td>128</td>
<td>%</td>
</tr>
<tr>
<td>129</td>
<td>&amp;</td>
</tr>
<tr>
<td>130</td>
<td>'</td>
</tr>
<tr>
<td>131</td>
<td>(</td>
</tr>
<tr>
<td>132</td>
<td>)</td>
</tr>
<tr>
<td>133</td>
<td>*</td>
</tr>
<tr>
<td>134</td>
<td>+</td>
</tr>
<tr>
<td>135</td>
<td>,</td>
</tr>
<tr>
<td>136</td>
<td>-</td>
</tr>
<tr>
<td>137</td>
<td>.</td>
</tr>
<tr>
<td>138</td>
<td>/</td>
</tr>
<tr>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>1</td>
</tr>
<tr>
<td>141</td>
<td>2</td>
</tr>
<tr>
<td>142</td>
<td>3</td>
</tr>
<tr>
<td>143</td>
<td>4</td>
</tr>
<tr>
<td>144</td>
<td>5</td>
</tr>
<tr>
<td>145</td>
<td>6</td>
</tr>
<tr>
<td>146</td>
<td>7</td>
</tr>
<tr>
<td>147</td>
<td>8</td>
</tr>
<tr>
<td>148</td>
<td>9</td>
</tr>
<tr>
<td>149</td>
<td>:</td>
</tr>
<tr>
<td>150</td>
<td>;</td>
</tr>
<tr>
<td>151</td>
<td>&lt;</td>
</tr>
<tr>
<td>152</td>
<td>=</td>
</tr>
<tr>
<td>153</td>
<td>&gt;</td>
</tr>
<tr>
<td>154</td>
<td>?</td>
</tr>
<tr>
<td>155</td>
<td>?</td>
</tr>
<tr>
<td>156</td>
<td>A</td>
</tr>
<tr>
<td>157</td>
<td>B</td>
</tr>
<tr>
<td>158</td>
<td>C</td>
</tr>
<tr>
<td>159</td>
<td>D</td>
</tr>
<tr>
<td>160</td>
<td>E</td>
</tr>
<tr>
<td>161</td>
<td>F</td>
</tr>
<tr>
<td>162</td>
<td>G</td>
</tr>
<tr>
<td>163</td>
<td>H</td>
</tr>
<tr>
<td>164</td>
<td>I</td>
</tr>
<tr>
<td>165</td>
<td>J</td>
</tr>
<tr>
<td>166</td>
<td>K</td>
</tr>
<tr>
<td>167</td>
<td>L</td>
</tr>
<tr>
<td>168</td>
<td>M</td>
</tr>
<tr>
<td>169</td>
<td>N</td>
</tr>
<tr>
<td>170</td>
<td>O</td>
</tr>
<tr>
<td>171</td>
<td>P</td>
</tr>
<tr>
<td>172</td>
<td>Q</td>
</tr>
<tr>
<td>173</td>
<td>R</td>
</tr>
<tr>
<td>174</td>
<td>S</td>
</tr>
<tr>
<td>175</td>
<td>T</td>
</tr>
<tr>
<td>176</td>
<td>U</td>
</tr>
<tr>
<td>177</td>
<td>V</td>
</tr>
<tr>
<td>178</td>
<td>W</td>
</tr>
<tr>
<td>179</td>
<td>X</td>
</tr>
<tr>
<td>180</td>
<td>Y</td>
</tr>
<tr>
<td>181</td>
<td>Z</td>
</tr>
<tr>
<td>182</td>
<td>[</td>
</tr>
<tr>
<td>183</td>
<td>]</td>
</tr>
<tr>
<td>184</td>
<td>^</td>
</tr>
<tr>
<td>185</td>
<td>_</td>
</tr>
<tr>
<td>186</td>
<td>`</td>
</tr>
<tr>
<td>187</td>
<td>~</td>
</tr>
<tr>
<td>188</td>
<td>@</td>
</tr>
<tr>
<td>189</td>
<td>#</td>
</tr>
<tr>
<td>190</td>
<td>$</td>
</tr>
<tr>
<td>191</td>
<td>%</td>
</tr>
<tr>
<td>192</td>
<td>&amp;</td>
</tr>
<tr>
<td>193</td>
<td>'</td>
</tr>
<tr>
<td>194</td>
<td>(</td>
</tr>
<tr>
<td>195</td>
<td>)</td>
</tr>
<tr>
<td>196</td>
<td>*</td>
</tr>
<tr>
<td>197</td>
<td>+</td>
</tr>
<tr>
<td>198</td>
<td>,</td>
</tr>
<tr>
<td>199</td>
<td>-</td>
</tr>
<tr>
<td>200</td>
<td>.</td>
</tr>
<tr>
<td>201</td>
<td>/</td>
</tr>
<tr>
<td>202</td>
<td>0</td>
</tr>
<tr>
<td>203</td>
<td>1</td>
</tr>
<tr>
<td>204</td>
<td>2</td>
</tr>
<tr>
<td>205</td>
<td>3</td>
</tr>
<tr>
<td>206</td>
<td>4</td>
</tr>
<tr>
<td>207</td>
<td>5</td>
</tr>
<tr>
<td>208</td>
<td>6</td>
</tr>
<tr>
<td>209</td>
<td>7</td>
</tr>
<tr>
<td>210</td>
<td>8</td>
</tr>
<tr>
<td>211</td>
<td>9</td>
</tr>
<tr>
<td>212</td>
<td>:</td>
</tr>
<tr>
<td>213</td>
<td>;</td>
</tr>
<tr>
<td>214</td>
<td>&lt;</td>
</tr>
<tr>
<td>215</td>
<td>=</td>
</tr>
<tr>
<td>216</td>
<td>&gt;</td>
</tr>
<tr>
<td>217</td>
<td>?</td>
</tr>
<tr>
<td>218</td>
<td>?</td>
</tr>
<tr>
<td>219</td>
<td>A</td>
</tr>
<tr>
<td>220</td>
<td>B</td>
</tr>
<tr>
<td>221</td>
<td>C</td>
</tr>
<tr>
<td>222</td>
<td>D</td>
</tr>
<tr>
<td>223</td>
<td>E</td>
</tr>
<tr>
<td>224</td>
<td>F</td>
</tr>
<tr>
<td>225</td>
<td>G</td>
</tr>
<tr>
<td>226</td>
<td>H</td>
</tr>
<tr>
<td>227</td>
<td>I</td>
</tr>
<tr>
<td>228</td>
<td>J</td>
</tr>
<tr>
<td>229</td>
<td>K</td>
</tr>
<tr>
<td>230</td>
<td>L</td>
</tr>
<tr>
<td>231</td>
<td>M</td>
</tr>
<tr>
<td>232</td>
<td>N</td>
</tr>
<tr>
<td>233</td>
<td>O</td>
</tr>
<tr>
<td>234</td>
<td>P</td>
</tr>
<tr>
<td>235</td>
<td>Q</td>
</tr>
<tr>
<td>236</td>
<td>R</td>
</tr>
<tr>
<td>237</td>
<td>S</td>
</tr>
<tr>
<td>238</td>
<td>T</td>
</tr>
<tr>
<td>239</td>
<td>U</td>
</tr>
<tr>
<td>240</td>
<td>V</td>
</tr>
<tr>
<td>241</td>
<td>W</td>
</tr>
<tr>
<td>242</td>
<td>X</td>
</tr>
<tr>
<td>243</td>
<td>Y</td>
</tr>
<tr>
<td>244</td>
<td>Z</td>
</tr>
<tr>
<td>245</td>
<td>[</td>
</tr>
<tr>
<td>246</td>
<td>]</td>
</tr>
<tr>
<td>247</td>
<td>^</td>
</tr>
<tr>
<td>248</td>
<td>_</td>
</tr>
<tr>
<td>249</td>
<td>`</td>
</tr>
<tr>
<td>250</td>
<td>~</td>
</tr>
<tr>
<td>251</td>
<td>@</td>
</tr>
<tr>
<td>252</td>
<td>#</td>
</tr>
<tr>
<td>253</td>
<td>$</td>
</tr>
<tr>
<td>254</td>
<td>%</td>
</tr>
<tr>
<td>255</td>
<td>&amp;</td>
</tr>
<tr>
<td>256</td>
<td>'</td>
</tr>
<tr>
<td>257</td>
<td>(</td>
</tr>
<tr>
<td>258</td>
<td>)</td>
</tr>
<tr>
<td>259</td>
<td>*</td>
</tr>
<tr>
<td>260</td>
<td>+</td>
</tr>
<tr>
<td>261</td>
<td>,</td>
</tr>
<tr>
<td>262</td>
<td>-</td>
</tr>
<tr>
<td>263</td>
<td>.</td>
</tr>
<tr>
<td>264</td>
<td>/</td>
</tr>
</tbody>
</table>
INDEX

(LIB)<1>BRANCH  28,29
(LIB)<1>SWITCH  7,19
(LIB)<1>TRUNK  27
(LIB)<1>UPNODE  29
(LIB)<240>EDITOR  31
(LIB)<96>BUILD  17

ABORT  55
ADATE  61
ALPHA  53
AND  49
ANY  55
APPEAR  18,43
ARB  54
ARSIZE  50
ASCII  53

BACK  19,27,28,49
BEAMST  60
BEEF  32
BHELD  46
BIT  49
BOX  9,42
BREAK  18,43,55
BUTS  22,45
BUTTON  22,45
Button  20

CAMAC  2,31,48
CAPAB  36
CFIND  26,44
CHAIN  51
CHSIZE  42
CIRCLE  42
CLEAR  40
CLOSDL  61
CLOSE  56
COLOUR  45
COLUMN  46
COPY  51
CTIME  57
CURC  25,46
CURCON  24,48
CURL  25,47
CURSIZ  48
CYLENG  57

DATE  61
DAYTIM  46

DIAGRM  15,38
DICO  5,8,16
DIME  5,8,9
DISCON  58
Display  18
DLSTAT  61
DOT  39
DOTLEN  41
DOTTED  41

ENTERC  36
ERMESS  50
ERRLIN  50
ERROR  50
EVAL  53
EXECUTE  34

FAIL  55
FCOPY  61
FIND  53
FINDS  53
FIRST  51
FBUCC  34,46
FROMC  60

GETV  52
HBALL  24,47
HCURS  25,47
HELP  19,27,29,31,52
HISTO  15,40
HORIZO  41

IDEV  3
INBT  56
INDEX  61
INPC  56
INPUT  56
IOR  49

Keyboard  4
KILL  57
KNOB  22,44
Knob  22

LABEL  44
LDEF  32
LEGEND  6,20,45
Legends  29
LEGS 27,29
LEN 54
LINE 46
LISA 52
LISC 52
LISD 52
LISP 57
LISR 52
LISV 52
LOAD 32
LOC 50
LOCAL 61
MACHST 59
MAX 51
MCURS 25,48
MIN 51
MKNOB 23,44
MOVE 9,40
NEG 50
NEXT 28,49
NODAL 3,6,9,19,28,31,32
NODLIN 51
NOTANY 55
NUM 53
OCTANT 42
ODEV 5,6,7,9,18,40
OLD 32
OPEN 56
OPENDL 61
ORIGIN 8,9,40
OUTBT 56
OUTC 56
OUTPUT 56
PAGE1 18
PCOPY 43
PINXIT 18,46
Pixels 5
PLOT 39
POINT 41
POS 54
PROGS 27,29
PROGS(1) 29
PURGE 43
REPEAT 57
RESET 4,19
RMOVE 42
RPOS 54
RTAB 55
RTIME 58
RTRUN 32,57
RUN 28
RUNOFF 54
RVECT 42
RWDATA 57
SAVE 32
SCHEDL 57
SCREEN 12,37
SECTN 36
SETATTI 58
SHIFT 50
SIZE 53
SKNOB 44
SOLID 9,42
SORT 54
SPAN 55
STRARG 50
SUBS 53
SWITCH 7,19
SYSTYP 60
TAB 55
TEXT 42
THISP 27,28,49
TIME 58
Touch 19
Tracker-ball 23
TRUNK 19,27,28,31,49
TVCNCT 43
TX 29,31
UKSYST 61
USER 36
VALID 37
VBALL 24,47
VCURS 26,47
VECT 9,41
VERTIC 41
WAIT 34
WAIT-CYCLE 31
WHERIS 43
WINDOW 12,37
WSTK  60
XAXIS  40
XOFF  12,37
XSCAL  12,37
XYDOT  39
XYPLOT  39

YAXIS  40
YDOT  38
YHISTO  39
YOFF  12,37
YPLOT  38
YSCAL  12,37