A SCREEN BUILD
SOFTWARE PACKAGE

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Carolyn Owens
August, 1980
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>A. Available Screen Build Packages</td>
<td>3</td>
</tr>
<tr>
<td>B. Proposed In-House Screen-Build Package</td>
<td>6</td>
</tr>
<tr>
<td>II. PROBLEM DEFINITION</td>
<td>8</td>
</tr>
<tr>
<td>A. Current Method for Building CRT screens</td>
<td>9</td>
</tr>
<tr>
<td>B. Criteria</td>
<td>13</td>
</tr>
<tr>
<td>C. Restrictions</td>
<td>14</td>
</tr>
<tr>
<td>III. DESIGN ALTERNATIVES</td>
<td>15</td>
</tr>
<tr>
<td>A. Tables</td>
<td>15</td>
</tr>
<tr>
<td>B. Input</td>
<td>19</td>
</tr>
<tr>
<td>C. Input Processing</td>
<td>21</td>
</tr>
<tr>
<td>D. Timing</td>
<td>24</td>
</tr>
<tr>
<td>E. Output</td>
<td>28</td>
</tr>
<tr>
<td>F. Screen Access</td>
<td>32</td>
</tr>
<tr>
<td>G. Errors</td>
<td>35</td>
</tr>
<tr>
<td>H. Program Organization</td>
<td>36</td>
</tr>
<tr>
<td>IV. COMPLETION OF THE SCREEN-BUILD</td>
<td>39</td>
</tr>
<tr>
<td>A. Testing</td>
<td>39</td>
</tr>
<tr>
<td>B. Documentation</td>
<td>40</td>
</tr>
<tr>
<td>C. Implementation</td>
<td>42</td>
</tr>
<tr>
<td>D. Costs and Benefits</td>
<td>44</td>
</tr>
<tr>
<td>E. Suggestions for Further Research</td>
<td>45</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>46</td>
</tr>
<tr>
<td>APPENDIX A: User's Manual</td>
<td>47</td>
</tr>
<tr>
<td>APPENDIX B: Flow Charts</td>
<td>72</td>
</tr>
<tr>
<td>APPENDIX C: Program Listing</td>
<td>83</td>
</tr>
<tr>
<td>APPENDIX D: Tables</td>
<td>113</td>
</tr>
<tr>
<td>APPENDIX E: CRT Requirements</td>
<td>117</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Test Screen -- Production Screen Menu</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td>Example of a Typical Screen Generation Program</td>
<td>10</td>
</tr>
<tr>
<td>3.</td>
<td>Test Screen -- Production Ingot Bank Screen</td>
<td>41</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Video Codes</td>
<td>16</td>
</tr>
<tr>
<td>II. Cursor Positioning Table</td>
<td>17</td>
</tr>
<tr>
<td>III. Video Codes</td>
<td>26</td>
</tr>
<tr>
<td>IV. Disc Monitor Table</td>
<td>33</td>
</tr>
<tr>
<td>V. Error List</td>
<td>37</td>
</tr>
</tbody>
</table>
Chapter I

INTRODUCTION

A "screen" is an information form (business form, production form, accounting form, etc.) which is displayed on a CRT. Via the screen, information is displayed to a CRT operator, who can then alter or add to this information.

Many computer applications in business require the generation of CRT screens. For example, a production foreman may want to see the production schedule for the next shift. By entering a CRT keyboard command this production schedule screen can be displayed. (See Figure 1 for an example of a simple screen.)

A screen-build package is a set of programs designed to create these screens. Essentially, the package takes simple screen generation commands and translates them into the more complex code required to produce the desired CRT screen. The screen-build package enables the programmer, with little or no knowledge of the actual screen generation mechanics, to design and build a CRT screen separate from the program(s) which actually use the screen.

Without such a program, the programmer is forced to include the screen generation code in every program that uses that particular screen. This is a very tedious and time consuming process, which involves the use of hexadecimal ASCII
Figure 1: Test Screen - Production Menu Screen
code. If different types of CRTs are used, then the program must be rewritten for each type of CRT that will use it, as screen generation codes differ for every brand of CRT. Consequently, a good screen-build package also allows the programmer the freedom of designing screens without regard to the type or model of CRT used. Once the programmer has designed a screen for a particular CRT, it need not be redesigned for a different CRT.

Because of the tedious, repetitious, and often difficult coding process involved, the screen-build process is very prone to error. With the availability of a screen-build package the programmer can build screens quickly and easily. The programs which use the screens are independent of the screens and the CRTs which display the screen. Errors are also easily corrected since there is little problem in generating a new screen to replace one in error. Use of a screen-build package also adds consistency to programs and screens, since all screens are built by a standard method using a standard format.

A. Available Screen Build Packages

Current publications on the subject of available screen-build packages are limited in number. The reason is that most major hardware companies offer some type of screen-build software package for their clients. If a package is developed outside of a hardware manufacturer, it is probably developed as a company or group effort and designed for a specific pur-
pose. Therefore the result is seldom published. Of the available screen-build packages, a few are mentioned and discussed below. The first two examples are of hardware manufacturer screen-build packages which are currently on the market. The other two examples are of in-house packages developed specifically for particular installations.

1. IBM PACKAGE, 'SCREENER' - This package was developed specifically for use with the IBM's data base management system -- IMS. 'SCREENER' is designed for non-intelligent terminals (e.g. IBM's 3270 terminals), and for the larger IBM computers (e.g. IBM 370 computers) which can support IMS. This package allows the programmer two different modes of input: batch or TSO (Time Sharing Option). In the TSO mode, the user is prompted for the needed screen generation information, and input edit checking and corrections are handled in a conversational mode.

2. MODCOM PACKAGE, 'TSX' - The 'TSX' package has several functions besides screen generation. Some of these functions: timesharing and transaction processing, terminal spooling, and terminal usage statistics. The screen generator feature of this package is a non-resident utility program, which aids the programmer in the creation and maintenance of screens. It accepts screen images and converts them to virtual terminal codes. The program stores these codes and maintains a screen directory file. Then when a screen is referenced by name, it is automatically retrieved and displayed on the CRT screen. Because the screens are not
part of the application program, screen maintenance is simplified. The 'TSX' package was designed for the MODCOMP 'CLAS-SIC' with at least 128K words of memory available. The major difference between the MODCOMP 'TSX' package and the IBM 'SCREENER' package is that each was developed for use on their own respective hardware.

The following packages were in-house projects, developed specifically for ARMCO by ARMCO employees. These packages have never been published and therefore are not currently available for purchase as are the packages discussed above.

3. SCREEN-BUILD PROGRAM, 'SBLD' (Implemented on Westinghouse computers) - This package was developed specifically for use with the Westinghouse process computers currently operating at ARMCO, Middletown Works. The package is written in ASSEMBLER language and translates simplified input into CRT code which is then stored on disc. The package source input is usually in the form of punched cards and the output can be in any combination of the following: disc text file, CRT screen display, and printer. The core requirements of this package is approximately 3K bytes.

4. SCREEN PACKAGE (Implemented on DEC computers) - This screen generation package was designed for use with the DEC computers at ARMCO, Middletown Works. The main prerequisites for the package were: create CRT independence, improve programming standards, and encourage the use of screen headings and literals. To resolve the problem of CRT independence,
on-line tables, specific to the CRT, are used. This allows different CRT models to be substituted for one another without changes in the application programs. This package also performs edit checking. The total core space requirements of this screen generation package is approximately 14K, due to the on-line CRT tables and added features included in this package.

B. Proposed In-House Screen-Build Package (Implementation on MODCOMP computers)

As can be seen from the above discussion, most hardware manufacturers offer their own screen build packages to be used with their hardware. The 'TSX' package, which is offered by MODCOMP, meets the general requirements for a screen build package. However, it was developed only recently to be used on the newest and largest MODCOMP computer. The 'TSX' package uses around 15K words of storage space, which is too much to run efficiently on the smaller MODCOMP computers. Also, the 'TSX' package performs many other functions besides building screens, such as scheduling, time sharing, edit checking, etc. Many of these functions are not necessary at this time, and, therefore, would only use valuable space. Some of these functions, such as edit checking, may be included at a later date.

Due to the size and complexity of the MODCOMP 'TSX' package and the hardware dependency of the other packages, the user (Process Computing - ARNCO, Ashland Works) decided
that a small screen-build package is needed to be used with all the MODCOMP computers. The following chapters present a detailed discussion of this screen-build package.
Chapter II

PROBLEM DEFINITION

The screen-build package is particularly designed for the Process Control MODCOMP computers of ARMCO, Ashland Works. The Process Computing Department of ARMCO, Ashland Works is currently responsible for five MODCOMP installations, each employing from two to six people. Each installation is responsible for its own computers, files, programs, screens, etc. Hard-wire interfaces between these installations do not currently exist. Therefore, programs developed at one installation must be physically transported and implemented at other installations. Each installation is also responsible for the control of certain production processes, and the use of CRT information screens plays a major role in this control. Production operators depend on these screens to supply them with accurate production process information and also as a means of inputting new process information for computer analysis.

The CRTs which are currently used by the Process Control Department are OMRON 8025 and OMRON 8030 models. However, since other CRT brands may be incorporated at some future time the package has been designed to accommodate any brand of CRT.
A. Current Method for Building CRT Screens

Presently there are over twenty CRTs used by the Process Computing Department - ARMCO, Ashland Works, and several hundred programs which run on the Process Control NODCOMP computers and interface with the CRTs. The specific purpose of many of these programs is screen generation. The programmer must incorporate all of the FORTRAN 'WRITE' and 'FORMAT' statements that will generate the desired screen. Figure 2 is an example of a typical screen generation program. Notice that the CRT video codes are set to corresponding hexadecimal equivalent codes in the 'DATA' statement at the start of the program (e.g. PFF - protect format off, CUH - cursor home, etc.). (A complete list of video codes can be found in the Appendix.) The CRT processes these hex codes and then performs the specified function on the CRT screen. Therefore, every program which needs this screen must include this screen generation code and hexadecimal equivalent codes. This is a redundant and tedious coding process. If the screen is to be used only once or by just one program, this method of generating screens is still lacking in that it is a complicated method of building screens. Also, no standardized technique for generating screens is employed, which causes documentation and program maintenance problems. Although the screen-build programs may be similar to one another, each programmer is left on his or her own where screen building is concerned. Without a standardized format, problems may arise with maintenance of screens and screen operation programs.
Program S09

Dimension x(2)

Integer CRT, CSC, CUH, EPF, ETX, KBL, KBU, PFF, PFN, PFI,
1STX, VRN+, VBR, VDR, VRH, VNM, VUL

Data CSC, CUH, EPF, ETX, KBL, KBU, PFF, PFN, PFT, STX, VRN,
1VBR, VDP, VHR, VNM, VUL/Z1B4A, Z1B48, Z1B69, Z037F, Z1B45,
2Z1B46, Z1B58, Z1B57, Z1B67, Z027F, Z1B65, Z1B66, Z1B64,
3Z1B39, Z1B34, Z1B61/

9500 CRT=8

23 FORMAT[2A2]

100 FORMAT["ENTER NO. FOR SCREEN LISTING",2X,2A2," - "2A2]

200 FORMAT[2A2]

901 FORMAT["0",1,TURN CHANGE INFORMATION",12X,"10-PIT

1FILE CORRECTION")

902 FORMAT["1",2,PIT FIRING STATUS (1-20)",10X,"11-DRAW

1FILE CORRECTION")

903 FORMAT["2",3,PIT FIRING STATUS (21-40)",10X,"12-

1CHANGE RUNNING COUNT")

904 FORMAT["3",4,BANK SCREEN",24X,"13-BANK SCAN PROGRAM")

905 FORMAT["4",5-HEAT ENTRY SCREEN",18X,"14-PIT SCAN

1PROGRAM")

906 FORMAT["5",6-DRAWING SCREEN")

907 FORMAT["6",7-CHARGING SCREEN")

908 FORMAT["7",8-HEAT FILE CORRECTION")

909 FORMAT["8",9-MENU SCREEN")

990 FORMAT[2A2,1X,2A2]

2101 FORMAT[3A2]

Figure 2: Example of a typical screen generation program.
WRITE(CRT,2101) PFF, CUH, CSC
CALL_WAIT(20,0,X)
WRITE(CRT,100) STX, EPF, PFT, ETX
WRITE(CRT,990) EPF, PFT
WRITE(CRT,901)
WRITE(CRT,902)
WRITE(CRT,903)
WRITE(CRT,904)
WRITE(CRT,905)
WRITE(CRT,906)
WRITE(CRT,907)
WRITE(CRT,908)
WRITE(CRT,909)
WRITE(CRT,23) KBU, PFN
WRITE(CRT,200) NO
WRITE(CRT,23) PFF, KBL

Figure 2 continued.
In summary, problems associated with the present method of generating screens are:

1. The present method is very time consuming for the programmer. The programmer must design and build each screen within every program which uses the screen. This puts an additional work load on the programmer.

2. The present method involves using many FORTRAN 'WRITE' and 'FORMAT' statements for screen generation, requiring extensive coding within each program that generates a screen.

3. The present method is extremely tedious, due to the amount of coding involved and the care that must be taken to produce the correct screens.

4. The present method is error prone due to the amount of coding involved in each program. Also, care must be taken in working with the hex codes, since any carelessness in this area often results in incorrect screens being generated.

5. The present method produces CRT-dependent screens. Since each brand of CRT uses different hexadecimal video codes, and sometimes completely different code transmission formats, a screen generation program must be written for each brand of CRT. Therefore, the programmer must be familiar with each brand of CRT and its corresponding video codes, and other specifications. This in turn places an even greater workload on the programmer.

Because of the problems with the present method of generating screens, the programmer is not encouraged to experiment with new and more complex screen design. This
can be a problem when information clarity is sacrificed for simplicity. Therefore, more freedom in the screen design area is of benefit to the programmer.

B. Criteria

The following criteria and specifications are divided into two categories: primary criteria and secondary criteria. The primary criteria were specified by the users: the Process Computing Department -- ARMCO, Ashland Works. The secondary criteria were suggestions or specifications from the user, which would simplify the package and its use.

Primary Criteria

1. The screen-build package is to be written in FORTRAN and to be run on the MODCOMP Process Computers.
2. The screen-build package should generate screens which are compatible with any brand of CRT.

Secondary Criteria

1. The screen-build package should be easy to use and should simplify the screen design and screen generation process.
2. The ability to create more than one screen with each screen-build package run should be available.
3. The input to the screen-build package should be in punched card form.
4. The screen-build package should accomodate
screen changes, replacements, and deletions.

C. Restrictions

The following restrictions are due to MODCOMP or installation limitations.

1. Although the new MODCOMP computers are 128K word machines, the screen-build must also run on the smaller 64K word MODCOMP computers. Therefore, the package must not exceed 10K words.

2. Presently, there are at least five MODCOMP computers on which the screen-build package will be run. Since these computers are not currently tied in to one another, the screen-build package must be adaptable to all of these computers, and be in a physically transferrable form (e.g. punched cards).

3. Most of the MODCOMP computers are operating at a rate of 9600 baud, and the CRTs that are presently being used are designed to operate at a 4800 baud rate. Therefore, timing or padding characters must be sent to the CRT following many of the cursor and video commands. This gives the CRT the necessary time to process each command before another command is sent.
Chapter III

DESIGN ALTERNATIVES

Staying within the limits of the criteria and restrictions stated above, many alternatives are available for designing a screen-build package. Some of the alternatives discussed below are based on the design of other screen-build packages. Others are alternatives available at the MODCOMP installation at Armco Steel, such as hardware options and software packages. The following is a detailed discussion of the alternatives considered in the design of the screen-build package.

A. Tables

In order for the screen-build package to be CRT independent, some method was needed for adapting the package to different brands of CRTs. Most of the screen-build packages discussed above are table-driven packages. That is, for every type of CRT to be used, a code table is included in the program data. (See Table I for an example of a CRT video code table. In Table II, an example of a CRT cursor positioning table is presented.) When a CRT type is referenced, its corresponding code table is then accessed. The table driven package design is a logical means of fulfilling the CRT independency criterion.
<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Cursor/Pointer Up</td>
<td>VNN</td>
<td>Video Normal</td>
</tr>
<tr>
<td>CPD</td>
<td>Cursor/Pointer Down</td>
<td>LCF</td>
<td>Local Copy Off</td>
</tr>
<tr>
<td>CPR</td>
<td>Cursor/Pointer Right</td>
<td>LCO</td>
<td>Local Copy On</td>
</tr>
<tr>
<td>CPL</td>
<td>Cursor/Pointer Left</td>
<td>VUL</td>
<td>Video Underline</td>
</tr>
<tr>
<td>CPM</td>
<td>Cursor/Pointer Return</td>
<td>VDN</td>
<td>Video Dim</td>
</tr>
<tr>
<td>CUH</td>
<td>Cursor/Pointer Home</td>
<td>VDR</td>
<td>Dim Reverse</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear Line from Cursor</td>
<td>VBL</td>
<td>Video Blink</td>
</tr>
<tr>
<td>CHD</td>
<td>Character Delete</td>
<td>VBR</td>
<td>Video Blink Reverse</td>
</tr>
<tr>
<td>CHI</td>
<td>Character Insert</td>
<td>PFT</td>
<td>Protect Field - Tab</td>
</tr>
<tr>
<td>CIN</td>
<td>Character Insert On</td>
<td>SPF</td>
<td>Protect Field</td>
</tr>
<tr>
<td>CIF</td>
<td>Character Insert Off</td>
<td>EPF</td>
<td>End Protect Field</td>
</tr>
<tr>
<td>SCU</td>
<td>Scroll Up</td>
<td>KBL</td>
<td>Keyboard Lock</td>
</tr>
<tr>
<td>SCG</td>
<td>Scroll Down</td>
<td>KBU</td>
<td>Keyboard Unlock</td>
</tr>
<tr>
<td>NPG</td>
<td>Next Page</td>
<td>QUO</td>
<td>Quote Mode On</td>
</tr>
<tr>
<td>PPG</td>
<td>Previous Page</td>
<td>BEP</td>
<td>Set Beep Character</td>
</tr>
<tr>
<td>PFN</td>
<td>Protect Format On</td>
<td>STX</td>
<td>Start of Text</td>
</tr>
<tr>
<td>PFF</td>
<td>Protect Format Off</td>
<td>ETX</td>
<td>End of Text</td>
</tr>
<tr>
<td>HTS</td>
<td>Horizontal Tab Set</td>
<td>EPN</td>
<td>End Protect, Non-Alpha</td>
</tr>
<tr>
<td>HTC</td>
<td>Horizontal Tab Clear</td>
<td>UCP</td>
<td>Upper Case Only</td>
</tr>
<tr>
<td>VRV</td>
<td>Highlight on, Video Reverse</td>
<td>DEL</td>
<td>Delete</td>
</tr>
<tr>
<td>RCA</td>
<td>Read Cursor Address</td>
<td>VFF</td>
<td>Video Field Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SPA</td>
<td>Space</td>
</tr>
</tbody>
</table>

Table I: Video Codes
<table>
<thead>
<tr>
<th>Row and Column Number</th>
<th>Cursor Positioning Sequence</th>
<th>Row and Column Number</th>
<th>Cursor Positioning Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>@C</td>
<td>21</td>
<td>AE</td>
</tr>
<tr>
<td>01</td>
<td>@A</td>
<td>22</td>
<td>AF</td>
</tr>
<tr>
<td>02</td>
<td>@B</td>
<td>23</td>
<td>AG</td>
</tr>
<tr>
<td>03</td>
<td>@C</td>
<td>24</td>
<td>AH</td>
</tr>
<tr>
<td>04</td>
<td>@D</td>
<td>25</td>
<td>AI</td>
</tr>
<tr>
<td>05</td>
<td>@E</td>
<td>26</td>
<td>AJ</td>
</tr>
<tr>
<td>06</td>
<td>@F</td>
<td>27</td>
<td>AK</td>
</tr>
<tr>
<td>07</td>
<td>@G</td>
<td>28</td>
<td>AL</td>
</tr>
<tr>
<td>08</td>
<td>@H</td>
<td>29</td>
<td>AM</td>
</tr>
<tr>
<td>09</td>
<td>@I</td>
<td>30</td>
<td>AN</td>
</tr>
<tr>
<td>10</td>
<td>@J</td>
<td>31</td>
<td>AO</td>
</tr>
<tr>
<td>11</td>
<td>@K</td>
<td>32</td>
<td>B@</td>
</tr>
<tr>
<td>12</td>
<td>@L</td>
<td>33</td>
<td>BA</td>
</tr>
<tr>
<td>13</td>
<td>@M</td>
<td>34</td>
<td>BB</td>
</tr>
<tr>
<td>14</td>
<td>@N</td>
<td>35</td>
<td>BC</td>
</tr>
<tr>
<td>15</td>
<td>@O</td>
<td>36</td>
<td>BD</td>
</tr>
<tr>
<td>16</td>
<td>@Q</td>
<td>37</td>
<td>BE</td>
</tr>
<tr>
<td>17</td>
<td>AA</td>
<td>38</td>
<td>BF</td>
</tr>
<tr>
<td>18</td>
<td>AB</td>
<td>39</td>
<td>BG</td>
</tr>
<tr>
<td>19</td>
<td>AC</td>
<td>40</td>
<td>BH</td>
</tr>
</tbody>
</table>

**Table II: Cursor Positioning Table**
Within the design of the tables and their use, several alternatives were considered:

1. **Built-in tables.** The tables could be included as part of the main program in the screen-build package, using arrays and DATA statements. With this method, all of the tables would always be available to the user. However, this method requires a large amount of storage space, and only the tables corresponding to the particular CRT type used would actually be utilized. This means all of the other CRT tables would be occupying, thereby wasting, storage space. Since one of the restrictions was limited space, built-in tables were not utilized.

2. **Calculated cursor positioning tables.** Although some CRT cursor positioning tables lend themselves to being calculated, most CRT video code tables do not. The advantage of being able to calculate the tables is that very little storage space is used, since a table value is calculated instead of stored. Unfortunately, most CRT video code tables cannot be calculated. The screen-build package design was to be general enough to accommodate all brands of CRTs, so it was not feasible to calculate the tables.

3. **Input tables.** The alternative chosen is to read the specified CRT tables into arrays whenever the package is run. This allows the user to determine the necessary tables by simply specifying the type or brand of CRT on which the screen will be displayed. This method also saves space, since only the specified CRT tables are loaded into core.
One of the disadvantages of this method is that all of the CRT brand tables must be kept available to be read in as input. They must be physically located in the user's input deck or stored off-line. Also, some method of distinguishing between different brands of CRT tables must be established. However, these disadvantages are relatively minor, and since this method fulfilled all of the criteria, it was chosen as the design alternative.

In the actual screen-build package design, the user supplies the main program with the CRT type. The main program then calls an overlay (called 'TBLSUB') which searches for the correct tables to be read in. Each set of tables has a table header card which contains the CRT type. When the correct tables are found, they are read into the table arrays, and all other tables are then bypassed. (A more complete description of the overlaying procedure appears later in this chapter.)

B. Input

Excluding the table input discussed above, the user must input a sequence of screen-build commands describing the screen to be constructed. The program interprets these commands, generating the code that will actually produce the desired screen. In order to fulfill the criteria that the screen-build package be easy to use and simplify the screen generation process, the user input commands must be kept fairly simple and recognizable. Most of the currently available
screen-build packages accomplish this by using alphabetic characters or numeric input commands. Other screen-build packages are primarily designed for CRT input (i.e., the user actually creates the desired screen on the CRT using specified command characters and text fields). Since punched card input was specified, the CRT input alternative was ruled out.

1. **Numeric input.** With numeric input, the user is given a list of screen-build commands with a corresponding numeric value assigned to each command. The user simply looks down the list for the desired command and then enters the corresponding numeric value. The main program in the screen-build package contains a similar list to be searched for the corresponding command, which would then be executed. The drawback of this method is that the user must always have a copy of the command list, incorporating all of the screen-build commands. Approximately forty screen-build commands are necessary. The potential users in Process Control believed that such a list would be awkward to use, and would produce more programming errors than the other alternative. Therefore, the numeric input alternative was rejected.

2. **Alphabetic Character Input.** This method is similar to the numeric input method in that the user is given a list of screen-build commands. However, corresponding to these commands are alphabetic characters instead of numeric values. The major advantage in using alphabetic input characters is that these characters can be associated, in the user's mind, with the corresponding commands. They are in a sense abbre-
viations for the screen-build commands. Because of this, the alphabetic input method is the least confusing of the two methods considered.

Numeric values do play a part in the actual input commands, in that some commands require integer values such as row and column positions for the cursor positioning command. In such cases, the user supplies the screen-build program with these integer values and the program does the necessary interpretation.

The program was designed to accept free format input. That is, the programmer need not position the input commands in a specific column. This eases the programming task, and will likely reduce the frequency of input errors.

C. Input Processing

This section presents two design decisions concerning the processing of input commands: first, the use of overlays in place of subroutines, and second, the choice of table look-up techniques for searching the CRT input table arrays.

1. **Overlay vs. Subroutine.** The screen-build package was originally designed to process input commands using subroutines only. The advantage of using subroutines instead of overlays is that subroutines are loaded into main memory just once, then can be referenced by other routines at any time. However, subroutines occupy prime (i.e., core) storage space throughout the total execution of the screen-build package. This means the subroutine is taking up space even
when it is not being used.

In contrast, the overlay is stored off-line. When it is referenced by the program, it is then brought into core. When another overlay is referenced, it is brought into core, "overlaying" the previous one, and utilizing the same core space. In this way, core space is not wasted. The disadvantage in using the overlay is the time that is spent in loading in the overlays. Therefore, there is a time-space tradeoff in using overlays instead of subroutines.

Experimentation with different combinations of overlays and subroutines revealed that a large amount of space is saved, with minimal time increase, by overlaying only the larger subroutines. The small subroutines are left as such. Using this method, the total core utilized during the screen-build package execution was reduced by 2400 words. (Before overlaying, the package used 5700 words of main storage, and after overlaying it used only 3300 words.) The execution time is not greatly increased, and since one of the restrictions was space and core utilization, overlaying the larger routines is the best alternative.

2. Table Look-up Techniques. There are several look-up techniques commonly used for searching arrays. The hashing technique is one of the most popular, and complex, table look-up methods. However, hashing is really a technique to be used with large tables (i.e., 100 or more entries). Since most of the CRT tables have less than a hundred entries, it was decided to use a method better suited for small tables.
A binary search technique can be used when a pre-sorted array already exists. The binary search repeatedly divides the table in half and determines which half contains the desired value. This technique would lend itself to the CRT cursor positioning tables (see Table 2.1), since the rows and column values are in order (1 through 80). However, the CRT video tables (see Table 1) contain alphabetic codes which would have to be translated into their numeric values before any comparison or table halving could be performed. Due to this and to the fact that the video code tables are small (i.e., no greater than 40), the binary search is used only for searching the CRT cursor positioning tables.

Since the CRT video code tables contain a small number of entries, it is logical to search them sequentially. A sequential search (i.e., examining each entry until the correct one is found) can be a very slow process. But since the number of entries in the CRT video code tables is small, and no numeric conversions need to be performed (only a simple comparison is made), the sequential search technique is applied in searching the video code tables.

In summary, the binary search technique is used to search the CRT cursor positioning table for the correct row/column hexadecimal value, and the sequential search method is used to search the CRT video code table for the corresponding hexadecimal function value.
D. Timing

The problem of timing arises when the computer is transmitting at a faster baud rate than the CRT microprocessor can process and react. The Omron 8030 CRT model operates optimally at a rate of 4800 baud. However, the MODCOMP computers are currently transmitting at a rate of 9600 baud. As a result, the CRT cannot process some of the video codes in time to receive the next transmission, so the next transmission is lost. The remedy is to find out which function codes require the extra processing time, then transmit enough padding, ("timing") characters following these functions to ensure that the function code is completely processed before the next function code is transmitted.

Since the Omron 8030 model is designed to operate at or below a rate of 4800 baud, there is no accurate manufacturer information available for operation above 4800 baud. This meant the video function codes had to be tested to find out which ones required extra processing time.

To test the function codes, a small program was written for the sole purpose of filling a buffer with a function code and transmitting the filled buffer to the CRT. Characters were then sent at the end of the buffer transmission. If the characters appeared correctly on the CRT screen, then the function code was assumed to require no timing. If some of the characters were missing or altered, then timing was assumed to be necessary. In this case, the buffer was filled again with the function code, but one timing character was placed
between each code. This process was repeated, incrementing the timing characters each time by one, until the correct number of timing characters was discovered.

In the timing test program, the buffer was dimensioned as a 200 word integer array. The rationale is that the screen-build package currently transmits a 40 word buffer to the CRT, but the option exists to increase the size of this buffer. Under normal operating conditions, the buffer size will never be defined to be much above 100 words, as this approaches the physical buffer size. Consequently, no benefit would be derived from a larger buffer size. Defining the test buffer at 200 words allowed for almost any increase in the screen-build package output buffer that might occur. Filling the test buffer with a particular function code ensured that any extra timing required was specifically for that particular function code, and no other.

Two different timing characters could be sent following the function code transmission: the null character or the delete character. The null character is represented in hexadecimal code by a '00', while the delete is represented by a hexadecimal '7F'. Some CRTs process the null character and therefore treat it as a function code. The delete character, however, is essentially ignored by the micro-processor, and therefore is better suited for use as a timing character.

A summary of the number of timing characters required for each of the video function codes is presented in Table III. These results were obtained from the timing tests and test
<table>
<thead>
<tr>
<th>CODE</th>
<th>HEX</th>
<th>TIMING</th>
<th>CODE</th>
<th>HEX</th>
<th>TIMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1B41</td>
<td>0</td>
<td>VNM</td>
<td>1B34</td>
<td>1</td>
</tr>
<tr>
<td>CPD</td>
<td>1B42</td>
<td>0</td>
<td>LCF</td>
<td>1B3A</td>
<td>0</td>
</tr>
<tr>
<td>CPR</td>
<td>1B43</td>
<td>0</td>
<td>LCO</td>
<td>1B3B</td>
<td>0</td>
</tr>
<tr>
<td>CPL</td>
<td>1B44</td>
<td>0</td>
<td>VUL</td>
<td>1B51</td>
<td>1</td>
</tr>
<tr>
<td>CPM</td>
<td>1B47</td>
<td>0</td>
<td>VDM</td>
<td>1B63</td>
<td>1</td>
</tr>
<tr>
<td>CUH</td>
<td>1B48</td>
<td>0</td>
<td>VDR</td>
<td>1B64</td>
<td>1</td>
</tr>
<tr>
<td>CLC</td>
<td>1B4B</td>
<td>1</td>
<td>VBL</td>
<td>1B65</td>
<td>1</td>
</tr>
<tr>
<td>CHD</td>
<td>1B50</td>
<td>6</td>
<td>VBR</td>
<td>1B66</td>
<td>1</td>
</tr>
<tr>
<td>CHI</td>
<td>1B40</td>
<td>6</td>
<td>PFT</td>
<td>1B67</td>
<td>1</td>
</tr>
<tr>
<td>CIN</td>
<td>1B51</td>
<td>0</td>
<td>SPF</td>
<td>1B68</td>
<td>1</td>
</tr>
<tr>
<td>CIF</td>
<td>1B52</td>
<td>0</td>
<td>EPF</td>
<td>1B69</td>
<td>6</td>
</tr>
<tr>
<td>SCU</td>
<td>1B53</td>
<td>0</td>
<td>KBL</td>
<td>1B45</td>
<td>0</td>
</tr>
<tr>
<td>SCD</td>
<td>1B54</td>
<td>0</td>
<td>KBU</td>
<td>1B46</td>
<td>0</td>
</tr>
<tr>
<td>NPG</td>
<td>1B55</td>
<td>0</td>
<td>QUO</td>
<td>1B5A</td>
<td>0</td>
</tr>
<tr>
<td>PPG</td>
<td>1B56</td>
<td>0</td>
<td>BEP</td>
<td>1B59</td>
<td>3</td>
</tr>
<tr>
<td>PFN</td>
<td>1B57</td>
<td>1</td>
<td>STX</td>
<td>027F</td>
<td>0</td>
</tr>
<tr>
<td>PFF</td>
<td>1B58</td>
<td>0</td>
<td>ETX</td>
<td>037F</td>
<td>0</td>
</tr>
<tr>
<td>HTS</td>
<td>1B31</td>
<td>0</td>
<td>EPN</td>
<td>1B70</td>
<td>1</td>
</tr>
<tr>
<td>HTC</td>
<td>1B32</td>
<td>0</td>
<td>UCO</td>
<td>1B4E</td>
<td>0</td>
</tr>
<tr>
<td>VRV</td>
<td>1B33</td>
<td>1</td>
<td>DEL</td>
<td>7F7F</td>
<td>0</td>
</tr>
<tr>
<td>RCA</td>
<td>1B7C</td>
<td>2</td>
<td>VFF</td>
<td>1B62</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SPA</td>
<td>207F</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table III: Video Codes**
program discussed above. Each of these timing character values are included in the video code tables in the screen-build package. When the user enters a video function code, its timing character value is checked to see if the function requires any timing. If it does, then the required number of deletes are output following the function code. If the timing character value is zero, then only the function code is output. Using this method, the problem of timing was solved for most of the newer CRT brands.

A problem still remains for the older CRT models. For example, the Omron 8025 model is designed to operate at a baud rate of 2400 or less. A very extensive amount of timing is necessary to make this model compatible with a transmission rate of 9600 baud. In fact, the large amount of timing characters required makes this method of sending timing characters very impractical for use with the Omron 8025 model. Two alternatives are available for solving this problem. The first alternative is to interface the MODCOMP computers with the Omron 8025 models at a baud rate of 2400 or less. Since there are very few Omron 8025 models in operation, this amounts to only one MODCOMP computer that must transmit at this low baud rate. Also, there are plans to phase out all of the Omron 8025 models and replace them with the newer, faster processing CRT brands. Until this occurs, the alternative of transmitting at a rate of 2400 baud or less must be accepted.
E. Output

The design of the screen-build package output falls into three categories: choice of output devices, design of output code, and design of the disc files.

1. Output devices.

There are presently three types of output devices available for the screen-build package output. These are printers, CRTs, and disc files. Originally it was intended to use the disc files for the screen-build output. Then when the screen code was put to the disc file, it would be a simple matter to call the screen up on a CRT from the application programs. Disc output is still the main output device used by the screen-build package. However, to aid the user in debugging his program, the ability to output directly to the CRT and the printer was added to the screen-build package. In fact, the user can output either to the printer, or to the CRT, or to both, and not output to the disc file, if the user wants to verify a screen design. Once the screen design is verified, the user may then output the correct screen to the disc file. Any combination of the above three output devices may be requested by the user for the screen output. In addition, by making use of the ability to output to the CRT and printer first, the user avoids the possibility of storing an incorrect screen on the disc file.

2. Output code.

Two options were considered for the type of output code
to be generated by the screen-build package. The first and perhaps most obvious type of output code is the actual screen generation hex code. This code type, when transmitted directly to the CRT, produces the desired screen. There are several advantages to using this type of output code. It is a simple method, the package outputs the hex codes and text strings either to the CRT, printer, or disc file (or any combination of the three). If this code is stored on disc, a simple read program can output it to the CRT. There are no further steps and no translation or interpretation occurs after the code is output. Therefore, very little core space is required to output the code to the CRT screen. The main disadvantage of using this type of output code is that once the screen output code is stored on the disc file, no changes can be made. If the user wants to change a screen, the screen-build package must be run again, and a new screen built. This could be very inconvenient if screens were designed and generated for use with one type of CRT, but due to technical problems, another CRT type was substituted. In this case, the screens would have to be rebuilt, meaning the screen-build package would have to be run again using the different set of CRT tables, which outputs code compatible with the substituted CRT. This could become a real problem if the CRT types were switched during shifts when no programmers were available to rerun the screen-build package.

The second option is to output general type of output code. When this code is read from disc and transmitted to
the CRT, it goes through an interpretation process. The CRT tables are stored in core, so that the general output code is valid for any CRT type. The interpretation process would interpret this general code, using the CRT tables, and then output the resulting hex code to the CRT. Since the interpretation process is performed after the output code has been stored on disc, the screen-build package need not be reexecuted for use with different CRT brands. However, core space is required for storing the CRT tables. Depending on the number of CRT brands used, this storage can be quite extensive. Also, core storage is necessary for the program which interprets the general screen code. This interpretation program would in many ways be similar to the screen-build package in that it would contain search routines and output routines. Again, this could use up large amounts of core storage space, which is limited on the smaller MODCOMP computers.

The users (in Process Computing) chose the first option (i.e., to output the actual screen generation code and not the general code). As noted above, the use of general output code requires too much prime storage space. Also, predominantly one type of CRT, the Omron 8030, is presently used in the mill. As a result, it is unlikely that switching CRTs will cause problems. If new CRT types were introduced into the process control operation, then either similar replacements would be made, or else the inconvenience of re-generating the screens would have to be accepted. The Omron 8025 models currently being used are normally replaced by
other Omron 8025 models. However, if this were not the case, the output for both the Omron 8025 and 8030 models would be similar enough to be interchangeable, provided the transmission baud rate was adjusted accordingly, and the programmer designed the screen to be used on both CRT models (i.e., using codes common to both CRT models).

3. **Disc files.**

Storing the screen-build package output on disc files created the problem of how to assign a screen to a disc file location and give the user the ability to access it from the user programs. There were two options available.

The first possibility is to allow the user to designate the disc file location for storing the screen code. The user would know exactly where on disc the screen was located, and could include this location in the user programs. However, every user would need assigned disc space, and would have to include the location of this disc space in the screen-build package. Also, all of this user disc space would have to be maintained (i.e., obsolete screens would need to be deleted, and new disc space acquired when necessary, etc.) by the user. This would add to the programmer's work load instead of lightening it, so the user and the author decided that a better method was needed.

The second option is to designate a large area of disc space for storing all of the screen-build package output, then to keep a table showing what screens were stored in this area.
and where within the area they are actually located. Sub-
routines within the screen-build package could update this
table. This involves updating screen names and the associated
starting and ending record pointers. This "monitoring table"
also solves the problem of user access to the screens. By
keeping this table in a global common area and allowing the
user to give each screen a unique name, the user's program
can then pass this unique name to the library subroutine. The
subroutine then searches the monitor table for the screen
name and its location on disc. Using these location record
pointers, the library subroutine reads the screen code from
the disc, and outputs it to the CRT. In Table IV, an example
of a disc monitoring table is given.

A disadvantage in using this method is that the screen-
build package must maintain the disc space and monitoring
table. Therefore, another subroutine, which handles screen
deletions, was included in the package. The user can request
the screen-build package to delete a screen by supplying the
delete input code and the screen name. This facilitates
correcting or changing screens already stored on disc, and
deleting obsolete screens to make room for new ones. This
subroutine also updates the disc monitoring table after any
screen deletions take place.

F. Screen Access

Since the screen-build package output is stored on a
disc file, some means of accessing this file must be avail-
<table>
<thead>
<tr>
<th>SCREEN NAME</th>
<th>NO.</th>
<th>STARTING RECORD NO.</th>
<th>ENDING RECORD NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCR1</td>
<td>1</td>
<td>000</td>
<td>007</td>
</tr>
<tr>
<td>SCR2</td>
<td>2</td>
<td>008</td>
<td>012</td>
</tr>
<tr>
<td>SCR3</td>
<td>3</td>
<td>013</td>
<td>017</td>
</tr>
<tr>
<td>SCR4</td>
<td>4</td>
<td>018</td>
<td>025</td>
</tr>
<tr>
<td>SCR5</td>
<td>5</td>
<td>026</td>
<td>030</td>
</tr>
</tbody>
</table>

Table IV: Disc Monitor Table
able to the user. The access routine could be designed either as a complete, separate program, or as a user library subroutine.

One advantage in using a separate program to access the screen disc file is that the (access) program is complete within itself. The application program need not include any of the screen access information and thereby is simplified. The disadvantage of using a program to access the screen disc file is that it must be a core resident program. Since minimum core usage was a specified criteria, this alternative was rejected.

Using a subroutine to access the screen disc file has the advantage of being a user library subroutine, which can be stored off-line, saving core space. However, the user must include the necessary dimension statements and subroutine parameters within the user program. This means more work for the programmer. This information amounts to only a few statements that must be included in the user program in order to call the access routine. Because of this small number of statements, and because the use of the subroutine saves resident core space, this method was chosen to access the screen disc file.

To call the screen access subroutine, the user includes within the main program the global common definition statement and its corresponding dimension statements. Also, the user must specify the CRT device number for the screen output and the screen name which will be output. Finally, he must
indicate whether or not the access program should clear the screen before screen transmission begins. These three parameters are passed to the access program in the subroutine call statement. For exact specifications on using the screen access subroutine, refer to the user's guide included in the appendices.

G. Errors

Although the screen-build package is designed to be easily used, some errors in user input are bound to occur. Therefore, a good error diagnostic routine had to be included in the screen-build package.

Within the error routine a decision had to be made concerning the action to be taken when input errors occur. Should the screen-build package run be terminated due to an error or should the screen-build process terminate but error checking continue? It was decided that only a severe error should cause the package run to terminate, otherwise a diagnostic message should be printed and an error checking mode entered. In this mode, the screen-build process would cease, but syntax error checking would continue and diagnostic messages would be printed. This type of action allows the programmer to find most of the input errors in the first run of the screen-build package.

After examining the types of errors that can occur, it was found that there was no reason for terminating the screen-build package run. All errors could be diagnosed,
and error checking mode entered, without harm either to the package, existing screens, or the operating system. To ensure this, when an error is encountered, the screen building process stops, the error message is printed, and the screen name and record pointers in the disc monitor table are set to a blank and zeros, respectively (provided the screen is being stored on the disc file). In Table V, the list of possible errors that may occur, and their corresponding error messages, is shown. Additions and changes to this list can be made easily, whenever maintenance requires.

H. Program Organization

The screen-build package could have been designed as one long program with subroutines performing repetitious functions. This method of program organization is best for situations requiring short, uncomplicated programs. Since it was known that the program would be long, and would perform fairly complicated tasks, it was designed instead as a group of modules. This method is known as block programming, and is the fundamental design technique used in the screen-build package. Block programming is the construction of a program as a collection of several independently designed components that contain their own set of properties. Pollack and Sterling (4) discuss the advantages of using this technique in program design. For instance, since the program consists of a group of modules, each module can be tested and checked separately. This allows for less testing and debugging upon
<table>
<thead>
<tr>
<th>ERROR NO.</th>
<th>ERROR MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DISC IS FULL</td>
</tr>
<tr>
<td>2</td>
<td>SCREEN NAME IS NOT UNIQUE</td>
</tr>
<tr>
<td>3</td>
<td>(/S) CARD FOUND WITHOUT A PRECEDING (/C...DSC) CARD</td>
</tr>
<tr>
<td>4</td>
<td>MAXIMUM NUMBER OF SCREENS EXCEEDED</td>
</tr>
<tr>
<td>5</td>
<td>CORRECT TABLE NOT FOUND</td>
</tr>
<tr>
<td>6</td>
<td>END OF TEXT STRING NOT FOUND</td>
</tr>
<tr>
<td>7</td>
<td>REPEAT LIMIT EXCEEDED</td>
</tr>
<tr>
<td>8</td>
<td>CODE NOT FOUND</td>
</tr>
<tr>
<td>9</td>
<td>ILLEGAL INPUT VALUE</td>
</tr>
<tr>
<td>10</td>
<td>NO NUMERIC VALUE FOUND ON (/=) CARD</td>
</tr>
<tr>
<td>11</td>
<td>A (/F) CARD WAS NEVER FOUND FOLLOWING A (/R) CARD</td>
</tr>
<tr>
<td>12</td>
<td>CODE TABLE MISSING</td>
</tr>
<tr>
<td>13</td>
<td>DISC IS SET ON WITHOUT A SCREEN NAME</td>
</tr>
<tr>
<td>14</td>
<td>MISSING (/B) CARD</td>
</tr>
<tr>
<td>15</td>
<td>NAME OF SCREEN TO BE DELETED WAS NOT FOUND</td>
</tr>
<tr>
<td>16</td>
<td>(/D) CARD ENCOUNTERED DURING A BUILD</td>
</tr>
<tr>
<td>17</td>
<td>INVALID CURSOR POSITION</td>
</tr>
</tbody>
</table>

Table V: Error list
completion of the program. Also, debugging the small modules individually is easier than debugging the total program upon its completion. Another major advantage of block programming is that, since the modules are separate blocks, they can be used again and again, even by other programs and packages. The modules can be stored in some type of user's subroutine library available for use by any program requiring them.
Chapter IV

COMPLETION OF THE SCREEN-BUILD PACKAGE

Upon completion of the screen-build package programs, several additional tasks were performed to prepare it for use. The package was tested as a single entity and checked for loose ends. It was also documented (i.e., system flow charts, user's guide, etc.) and implemented at other MODCOMP installations.

A. Testing

The major portion of the screen-build package testing was performed during the design of the package, by using block programming, and testing each module individually (as discussed in section H of Chapter III). Therefore, when the package had been completed, it was tested as a whole unit.

The package was originally designed to stop building the screen when an error was encountered, but to continue syntax checking until the end of the user input had been reached. Therefore, the first of these tests consisted of trying to abend the program, that is, trying to make the program stop processing correctly or completely, without proper warning to the user. This was done by including wrong codes and out of sequence cards and commands as input
to the screen-build package, and then checking the error messages to see how well they corresponded to the actual errors. These tests brought about some revisions in the screen-build package, such as including more checks for out of sequence commands. Also, additional error messages were added to the error message list, and some portions of the package were rewritten to adjust for errors that had not been originally foreseen.

The second area of testing consisted of building more screens. During the design of the screen-build package, simple screen displays were used to test the package. In the final tests, more complicated screens and screen variations were built. (See Figure 3 for illustrations of test screens.) As a result of these tests, an additional code was included in the video code tables. This code allows the user to move the cursor horizontally, one space at a time. Without this code, the programmer would have to calculate the row and column positions for the cursor move, even if the cursor was to be moved only a few spaces. With this code included in the tables, the test screens were rebuilt using less input commands, and package testing was completed.

B. Documentation

Two types of documentation were included during the package design and development stages. These are the basic module flowcharts and program logic comments. When the screen-build package was completed, the flow-charts were
<table>
<thead>
<tr>
<th>PIT NO.</th>
<th>TEMPERATURE</th>
<th>CURRENT SET POINT</th>
<th>FUEL TYPE</th>
<th>FLOW</th>
<th>CHARGED TIME</th>
<th>EXPECTED ROLL TIME</th>
<th>EXPECTED SOAK TIME</th>
<th>TIME CHARGED</th>
</tr>
</thead>
</table>

Figure 3: Test Screen — Production Ingot Bank screen
redrawn to show how the modules interface with one another. The flowcharts are presented in Appendix B. The comments throughout the program were for the most part left unchanged, except when program revisions made it necessary to add or change comments. The listing of the commented main program, overlays, and subroutines comprising the screen-build package can be found in Appendix D.

The remaining documentation is a user's guide, to assist the programmer in using the screen-build package. The user's guide consists of three areas. The first deals with using the screen-build package, types of input, general program description, and input command order. The second major area is concerned with the display of screens, after they are built and stored on disc. This section includes an example of a simple application program used to display a screen (including the required JCL). The third section, added to the screen-build package user's manual after the package had been implemented on another MODCOMP installation, deals with program changes, storage requirements, global common requirements, and other details necessary for successful package implementation. The screen-build package user's manual is presented in Appendix A.

C. Implementation

The screen-build package was developed and implemented on a small (64K word) MODCOMP computer. Although the program was designed to be easily implemented on the different Armco
MODCOMP installations, some problems arose when the package was implemented on the larger (128K) MODCOMP Classic computers. This was due to installation differences and to different operating systems. The only major changes made to the program were in the JCL. After making these changes as well as the correct file assignments and storage allocations, the package was tested using the test screens mentioned above. One problem arose during this test phase, due to installation software differences. When a screen was sent to the CRT, an automatic line feed occurred at the end of every buffer transmission (or WRITE statement). This was undesirable in that screen lines were not positioned correctly on the CRT screen, resulting in incorrect screen display. The problem was solved by setting the bits which control this carriage return function. Once this function was disabled, the screens could be successfully transmitted to the CRT, and package implementation was complete.

Due to the fact that most Armco MODCOMP installations employ only a small number of personnel (usually from two to six people per installation), much of the user training took place on a personal basis during the package implementation. Users' manuals were supplied, and users were encouraged to read them and then experiment with the actual building of screens during the package test phase at each installation. This was found to be a good training method, in that any questions or ambiguities could be cleared up immediately.
D. Costs and Benefits

Development costs for the screen-build package were calculated based on a fixed personnel cost figure of $2750 per man-month (together with the number of man-months spent developing the screen-build package). The fixed personnel cost figure of $2750 per man-month is based on the average salaries and benefit costs for the Armco process computing personnel. Approximately five man-months were spent on the development of the package. This figure includes the technical support assistance received throughout the development and testing states. Personnel costs were the only costs used in this calculation, since currently no system for computer cost allocation exists in any of the MODCOMP installations. Therefore, the total package development costs were approximately $15,000.

The maintenance costs are estimated to be approximately $2750 per year. This figure is based on the package requiring no more than one man-month per year for maintenance, and the fixed personnel costs of $2750 per man-month.

In addition to the screen-build package benefits previously discussed, it is estimated that using the screen-build package reduces the programmer's total coding time by 20%. This reduction is based on the amount of time it takes to build a screen without using this package versus building a screen using the package. Another tangible benefit derived from the development of the screen-build package is that the package lends itself to other applications, such as
alarm systems. The alarm message is built on disc using the screen-build package and displayed on the CRT when an alarm condition occurs. Similar systems that require CRT display messages stored on disc are also benefitting from using the screen-build package.

E. Suggestions for Further Research

The screen-build package was designed with CRT independence in mind. However the only CRTs available for package testing were the OMRON CRTs. The package should be tested using other CRT brands to see how well it provides CRT independence, and what package modifications must be made to accomplish this.

Another line of research might be to extend the package to include some on-line functions. Perhaps making the cursor positioning function an on-line task would be an aid to the programmer. This would allow the programmer an easy method of positioning the cursor for data inputs from a CRT operator. Along this same line, small on-line tasks could edit this input data. When an error was made, an error message would be returned to the screen. With these additional tasks included, the screen-build package would be a complete system for generating screens and monitoring screen inputs.
REFERENCES


## APPENDIX A
SCREEN BUILD PACKAGE - USERS' MANUAL

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>48</td>
</tr>
<tr>
<td>Program Description</td>
<td>48</td>
</tr>
<tr>
<td>Input</td>
<td>50</td>
</tr>
<tr>
<td>Input Card Order</td>
<td>56</td>
</tr>
<tr>
<td>Table Parameters</td>
<td>57</td>
</tr>
<tr>
<td>MODCOMP Installation</td>
<td></td>
</tr>
<tr>
<td>Files</td>
<td>62</td>
</tr>
<tr>
<td>Devices</td>
<td>63</td>
</tr>
<tr>
<td>Global Common</td>
<td>63</td>
</tr>
<tr>
<td>The Application Program and Screen Display</td>
<td>66</td>
</tr>
<tr>
<td>Subroutines and Overlays</td>
<td>69</td>
</tr>
</tbody>
</table>
The main purpose of the Screen Build Package is to supply the programmer with a tool that aids in the development of CRT screen layouts. The Screen Build Package generates a text file from simplified input codes supplied by the programmer and stores the text file on disc. The programmer will no longer have to code screen layout commands and will be able to display a screen by calling a supplied subroutine and passing this subroutine the screen name. In this way, the Screen Build Package should save the programmer tedious work and coding time. It should also reduce error common to screen generation and layouts.

Another important purpose of the Screen Build Package is to create screens that are not CRT dependent. Since screens will be built outside of the application programs, they will not be CRT dependent and can also be accessed by multiple programs.

Program Description

The Screen Build Package generates a text file on disc, which will build a screen when it is output to a CRT. This text file is given a four character name by the user, called a screen name. The output from the package may be displayed
in any combination of the following:

1. CRT - When the program is finished building a screen, the text file may be output immediately to the CRT enabling the user to see the actual screen layout.

2. Printer - At the end of a 'build', the text file may be dumped to the printer. What appears on the printer are the text strings and cursor positioning codes. The video codes will be ignored by the printer.

3. Disc - The text file may also be stored on disc to be displayed later from the user's program(s). It is recommended that a screen be stored on disc only when the programmer is satisfied with the output.

Once a text file (or screen) is stored on disc, the user's program can display the screen by calling a supplied subroutine and passing its screen name. The screen name is a four character name, unique to that screen, which the user supplies during the building of the screen. The screen text file will remain on disc until the programmer requests the Screen Build Package to delete it.

Package input will be in source input form with an 80 character maximum (e.g. punched cards, floppy disc, etc.), and is generally in free format, with the exception that the first two columns are used as 'keys' to the content of the card and at least one blank must separate codes.
Input

There are three types of input cards that will actually generate a screen. These are the following:

1. Video Codes Card - This card contains simple three letter video codes that are converted to a hex equivalent value. When the CRT processes these hex values, it performs the designated operation, (e.g. cursor home, video blink on, etc.). A complete list of available video codes can be found in Figure A.

   The format for this card is a free format, in that columns 1 - 80 may be used and there can be more than one code per record, provided the codes are separated by at least one blank. The codes must be contained in the list in Figure A or have been added by the user through an equivalence card (see discussion on equivalence card below.)

2. Cursor Positioning Card - This card contains the row and column cursor positions for a 24 x 80 CRT screen.

   The format for this card is a / (slash) in column 1 and an M in column 2. The row number and column number for the cursor position may appear anywhere in columns 3 - 80 with at least one blank separating them. The row numbers must fall between 1 and 24 and must precede the column number. The column number must fall between 1 and 80.
<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Cursor/Pointer Up</td>
<td>VNM</td>
<td>Video Normal</td>
</tr>
<tr>
<td>CPD</td>
<td>Cursor/Pointer Down</td>
<td>LCF</td>
<td>Local Copy Off</td>
</tr>
<tr>
<td>CPR</td>
<td>Cursor/Pointer Right</td>
<td>LCO</td>
<td>Local Copy On</td>
</tr>
<tr>
<td>CPL</td>
<td>Cursor/Pointer Left</td>
<td>VUL</td>
<td>Video Underline</td>
</tr>
<tr>
<td>CPM</td>
<td>Cursor/Pointer Return</td>
<td>VDM</td>
<td>Video Dim</td>
</tr>
<tr>
<td>CUH</td>
<td>Cursor/Pointer Home</td>
<td>VDR</td>
<td>Dim Reverse</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear Line from Cursor</td>
<td>VBL</td>
<td>Video Blink</td>
</tr>
<tr>
<td>CHD</td>
<td>Character Delete</td>
<td>VBR</td>
<td>Video Blink Reverse</td>
</tr>
<tr>
<td>CHI</td>
<td>Character Insert</td>
<td>PFT</td>
<td>Protect Field - Tab</td>
</tr>
<tr>
<td>CIN</td>
<td>Character Insert On</td>
<td>SPF</td>
<td>Protect Field</td>
</tr>
<tr>
<td>CIF</td>
<td>Character Insert Off</td>
<td>EPF</td>
<td>End Protect Field</td>
</tr>
<tr>
<td>SCU</td>
<td>Scroll Up</td>
<td>KBL</td>
<td>Keyboard Lock</td>
</tr>
<tr>
<td>SCD</td>
<td>Scroll Down</td>
<td>KBU</td>
<td>Keyboard Unlock</td>
</tr>
<tr>
<td>NPG</td>
<td>Next Page</td>
<td>QUO</td>
<td>Quote Mode On</td>
</tr>
<tr>
<td>PPG</td>
<td>Previous Page</td>
<td>BEP</td>
<td>Set Beep Character</td>
</tr>
<tr>
<td>PFN</td>
<td>Protect Format On</td>
<td>STX</td>
<td>Start of Text</td>
</tr>
<tr>
<td>PFF</td>
<td>Protect Format Off</td>
<td>ETX</td>
<td>End of Text</td>
</tr>
<tr>
<td>HTS</td>
<td>Horizontal Tab Set</td>
<td>EPN</td>
<td>End Protect, Non-Alpha</td>
</tr>
<tr>
<td>HTC</td>
<td>Horizontal Tab Clear</td>
<td>UCO</td>
<td>Upper Case Only</td>
</tr>
<tr>
<td>VRV</td>
<td>Highlight on, Video Reverse</td>
<td>DEL</td>
<td>Delete</td>
</tr>
<tr>
<td>RCA</td>
<td>Read Cursor Address</td>
<td>VFF</td>
<td>Video Field Off</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SPA</td>
<td>Space</td>
</tr>
</tbody>
</table>

Figure A: Video Codes.
3. **Text String Card** - This card contains any text string which the user wants to output to the CRT. The format for this card is a $ (dollar sign) in column 1 for the upper case letters or a ¢ (cents sign) for lower case letters, followed by the text string. A $ or ¢ must appear at the end of the text string, respectively.

The above cards are used for the output of the screen text field. The following card types are used as control cards for the Screen Build Package.

1. **CRT type Card (Mandatory)** - This card supplies the package with the CRT type and the CRT logical device number. It must be the first user supplied input card and it must be placed preceding the tables. The program then decides which tables to input, based on the information found on this card.

   The format is the CRT type appears in columns 1 - 3 in integer form, therefore right justified (e.g. for the Omron 8030 a '30' is placed in columns 2 and 3). In columns 4 - 80, the CRT logical device number should appear (e.g. free format).

2. **Delete Card (Optional)** - This disc inputs the command to delete a screen from the disc file and the name of the screen to be deleted. The delete card(s) must be placed following the tables but **before** any 'screen
build/ input.

The format for this card is a / (slash) in column 1 and a D in column 2. The four character screen name may then appear anywhere in columns 3 - 80. For every screen to be deleted, one delete card must be used (e.g. to delete three screens, three delete cards must be used) except if all screens are to be deleted, then a // (two slashes) replaces the four character screen name on a delete card. Care should be exercised when using any delete card, especially one containing a //.

3. Build Card (Mandatory for building screens) - This card tells the package to start building a screen. It must always follow the delete card (s) if the user wishes to build a screen. If no delete cards are present, the build card should immediately follow the tables. It can also be used when the programmer wishes to build more than one screen. Then the build card is placed between the end of one screen build input and the start of the new screen build input.

The format is simply a / (slash) in column 1 and a B in column 2.

4. Device Control Card (Mandatory for building screens) - This card has the function of telling the program which output devices to use. Any combination of the following:

A. CRT ('CRT')
B. Printer ('TYP')

C. disc ('DSC')

A device control card must be used for each device to be used. These cards should be placed at the beginning of the first screen input.

The format is a / (slash) in column 1 and a C in column 2. In columns 3 - 80, the device name code (i.e. CRT, TYP, or DSC) is placed.

5. Screen Name Card (Optional) - This card inputs the four character name of the screen being built. The name must be unique to the screen name file and a 'DSC' (disc) device control card must precede the screen name card. The screen name card must follow the device control cards for the first screen input and it must follow the build card in the successive screen inputs.

The format for this card is a / (slash) in column 1 and an S in column 2 followed by the four character screen name in columns 3 - 80 (free format).

6. Repeat Card (Optional) - This card tells the package to output a group of codes a specified number of times. The code to be repeated appears on the cards following a repeat card and may consist of any screen generation code (i.e. video codes, cursor positioning, and text strings).

The format for this card is a / (slash) in column 1 and an R in column 2. The repeat number follows in
columns 3 - 80 (free format).

7. Finish Card (Must be used with Repeat Card) - This card tells the package that the end of the group of codes to be repeated has been reached. The format is a / (slash) in column 1 and an F in column 2.

8. Equivalence Card (Optional) - This card tells the package that the video code tables are to be modified. Following this card are cards containing the additional video codes and their hex equivalent.

The format for the equivalence card is a / (slash) in column 1 and an = (equals sign) in column 2, followed by the number of codes to be added to the video codes table in column 3 - 80 (free format).

The cards containing the video codes (one video code per card) are in the same format as the video code table entries. This is:

Columns 1 - 4 video code name - A4 format (real - so must be left justified)

Columns 5 - 9 hex equivalent - Z4 format

Columns 10 - 13 Timing - I4 format (integer - so right justified)

9. Comment Card (Optional) - This card is used for program documentation only and is printed as a comment.

The format is a C in column 1 followed by any
comment the user wishes to make (free format).

10. End Card (Mandatory) - This card tells the screen build package that the end of the programmer input has been reached.

   The format is a / (slash) in column 1 and an E in column 2.

Input Card Order

In the above discussion for each input card, the positioning of the card is mentioned if the card's relative position is important. The following rules will help clarify input card positioning.

1. The CRT type card must be the first user input card and it must be placed immediately preceding the tables.

2. The delete card (s) must immediately follow the tables if any screens are to be deleted.

3. The build card must follow the delete card (s) or the tables if no delete cards are present.

4. The device control cards should follow directly after the first build card in order to set the output devices on.

5. The screen name card should follow the device control cards when the screen is to be stored on disc, and otherwise, it should be omitted.

6. The equivalence card must precede any use of
additional video codes.

7. The end card must be the last card in the user input deck and should not be placed immediately after a build card.

8. The repeat card must have a finish card at the end of the group of repeat codes.

When the above rules are not followed, the Screen Build Package will stop building screens and error messages will be printed. Screens that have already been built and stored on disc are not altered; but a screen currently being built when an error occurs will not be completed or stored on disc. The program will continue error checking until the end of the input is reached.

Input cards that do not have specified positions (see the discussion) can be placed anywhere between two build cards or a build card and an end card. The comment card can be placed anywhere in the input deck following the tables. Figure B shows an example of input to the Screen Build Package to build a simple menu screen.

Table Parameters

Code Tables - Since the Screen Build Package is a table-driven package, the addition of tables is a necessity for CRT independence. When tables are added to the package they should follow the prescribed format (see above discussion on equivalence card). If this is not possible then adjustments to
the format statements and the programs must be made. This involves adjusting the read statement and corresponding format statement for modifying code tables in the main program and also adjusting these statements in the 'TBLSUB' overlay. The cursor positioning tables must precede their corresponding video codes tables and they should be grouped in pairs according to their CRT type (e.g. both tables for the 8030 CRT should be placed together). It should also be noted that all tables should precede the /* card which marks the end of the tables section.

Marking the start of every table, there must be a table header card which supplies the package with the table type (i.e., video 'CODE' or cursor positioning/address 'ADD'), CRT type (e.g. '30' for Omron 8030), and length of the table (or total number of entries in the table). The format for the table header card is as follows:

Columns 1 - 4 table type ('code' or 'add') - A4 (must be left justified)

Columns 5 - 8 CRT type - I4 format (right justify)

Columns 9 - 11 table length - I3 format (right justify)

It should also be noted that if additions or deletions are made to existing tables, the table length on the header card must be adjusted accordingly.

Within the main program, the table arrays, which are 'ACODE' (code name) and 'ICODE' (hex equivalent) and 'ITIME' (timing) for the video tables and two dimensional 'LOC' array for the positioning tables, must be dimensioned to accomodate
Figure B: Sample Screen Build Package input to build a menu screen.
the largest tables that will be stored in them. Therefore, if tables are added that are larger than the existing tables, the above arrays may have to be re-dimensioned.

Disc Monitor Tables - The disc monitor tables keep track of the total number of screens, the screen names, and the starting and ending record numbers for each screen. If the total allowable number of screens is to be increased or decreased, the arrays 'SCRNAM', and the three dimensional array 'MONITR' must be re-dimensioned in both the main program and in the overlays 'DSCSUB' nad 'CMPSUB'. Also, the equivalence statements which equivalence 'SCRNAM' and 'MONITR' with global common ('IBG' array) should also be checked to make sure global common has not been exceeded and that the relationships are still valid.

It should also be noted that if the record number or the define file statement is altered, the variable 'LREC' should be set to the new record number.
MODCOMP INSTALLATION

Files

A define file statement must appear in the main program and all of the define file parameters must be assigned to corresponding variable names.

The general form of the define file statement is:
Define File U (m, r, f, v)

Where

U is the logical unit number used to reference the file;

m specifies the number of records in the file;

r specifies the maximum record size;

f specifies that records will be transmitted either with or without format control;

v is the associated variable.

An example of a define file statement is:
Define File 30 (50, 40, V, IVAR)

Then in the main program, the parameters assigned to the variable names are:

IDISC = 30 (where 30 is the file number
LREC = 50 (where 50 is the number of records in the file)

IQUIT = 40 (where 40 is the length of the record)

While the above parameters may vary depending on installation requirements, the parameters 'U' and 'IVAR' will
always be the same in the define file statement and are not assigned to any variable names.

It should also be noted that the array 'IOUT' must be dimensioned equal to the maximum record size defined in the define file statement 'IOUT' is dimensioned in the main program and in the IOSUB and WRTSUB subroutines.

Devices

The logical input and output devices must also be assigned to variables in the main program. The input device number is assigned to the variable 'IUNIT'.

   e.g. IUNIT = 5

The logical device number for the printer must also be assigned to the variable 'ITYP':

   e.g. ITYP = 6

The 'IUNIT and 'ITYP' variables already exist in the program and need only be changed to the correct logical device numbers.

Global Common

Since the screens are stored on a disc file and must be accessed by the application programs, tables are stored in global common containing the names and locations of the screens stored on disc. The following is a representation of these tables.
SCREEN LOCATION TABLES

<table>
<thead>
<tr>
<th>Screen Name</th>
<th>Screen #</th>
<th>Starting Record #</th>
<th>Last Record #</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCRA</td>
<td>1</td>
<td>000</td>
<td>007</td>
</tr>
<tr>
<td>SCRB</td>
<td>2</td>
<td>008</td>
<td>011</td>
</tr>
<tr>
<td>SCRC</td>
<td>3</td>
<td>012</td>
<td>017</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A global common area must be set aside for these tables. The size of this area depends upon the requirements of the installation and the availability of global common space. Once this space is acquired, a common statement defining this space must appear in the main program.

```
e.g. COMMON/BGLOB/ABC/(768)
```

Following this statement must be an equivalence statement defining the screen name tables and the screen location on the global common area.

```
e.g. EQUIVALENCE (IBG,SCRNAM), (IBG(201),MONITR(1,1))
```

The array named 'SCRNAM' is the array containing the screen names and must be declared as REAL in the main program. The length of this array may vary depending on the number of screens that will be stored on the disc file.

```
e.g. REAL SCRNAM(100)
```

The above example will allow a hundred screen names to be stored in the 'SCRNAM' array.

The array named 'MONITR' is a two dimensional array containing the locations of the screens on the disc file. It must be declared as INTEGER in the main program. The length of the array must correspond with the length declared for
'SCRNAM' and there must be three columns declared for 'MONITR'.

  e.g. INTEGER MONITR(100,3)

Also, in the main program, the variable 'MNTR' must be set to the length of the 'MONITR' array.

  e.g. MNTR = 100

Two overlays must also contain these GLOBAL COMMON and EQUIVALENCE statements as well as the corresponding REAL and INTEGER statements that define 'SCRNAM' and 'MONITR' array. These overlays are the 'DSCSUB' overlay and the 'CMPSUB' overlay.

It should be noted that the screen build package already contains the statements discussed above. Therefore, these statements should be adjusted to fit installation requirements and not added to package as new statements.
THE APPLICATION PROGRAM AND SCREEN DISPLAY

To display a screen, the application program must include the same statement that was discussed above in the GLOBAL COMMON section. This is: COMMON/BGLOB/ABC(768) or comparable global common statement that is defined in the main program of the Screen Build Package.

The application program must also include a variable name containing the logical device number for the CRT where the screen will be displayed. The program must also include the name of the screen being displayed, and the indication for clearing the screen. The indicator for clearing the screen must be set to 1 if the programmer desires the CRT screen to be cleared and the cursor homed before the screen is displayed.

When all of the above information has been included in the application program, a call can be made to the library read subroutine 'READSC', which will locate the correct screen and display it on the designated CRT. The parameters which must be passed in the call are: the screen name, the CRT variable, the printer variable, and the clear screen option.

An example of a call used in an application program is:

```
CALL READSC(SNAME,ICRT,ICLEAR)
```

where:

SNAME is the screen name;

ICRT contains the logical CRT device number;
CSC contains the clear screen designation.

(NOTE: The above example shows the parameter order that must be followed in calling the READSC subroutine.)

For an example of a very simple application program with JCL, see Figure C.
Figure C: This application program will cause the CRT screen to be cleared and the screen named SCR1 will be displayed on the CRT with logical device number 8.
Besides the main program and the input tables, the Screen Build Package consists of seven overlays and five subroutines. The five subroutines, which are:

- IOSUB
- WRTSUB
- CHKSUB
- INISUB
- READSC

may be stored off-line in a user subroutine library to save load-in time. The seven overlays which are:

- ERRSUB
- DSCSUB
- PALSIB
- TBLSUB
- ADDSUB
- CDESUB
- CMPSUB

can be catalogued and then overlayed in the JCL according to installation requirements. It should be noted that this overlaying procedure differs from installation to installation and major differences exist between the MODCOMP MAX III and MAX IV operating systems. A functional flowchart of the Screen Build Package is shown in Figure D.

(Note: The subroutine READSC is not shown on this
chart since it is the subroutine that is called by the application program to display a screen on the CRT.)
APPENDIX B

FLOW CHARTS
MAIN PROGRAM

1. Initialize

2. Read 'CRT TYPE' card

3. Interpret 'CRT TYPE' card

   - 'ADD3UB'
   - 'TBLSUB'

4. Error? (Yes -> 199, No)

5. Set record pointers

6. Initialize main variables

7. Read and write input card

8. Process text string

   - 'PHKSUB'

9. Error? (Yes -> 199, No)

10. Control card?

    - Yes -> 100
    - No

11. Comment card?

    - Yes -> 10
    - No

12. Not a text string?

    - Yes -> 50
    - No

13. Return to 10
MAIN PROGRAM

105

SET EQUIVALENCE CARD COUNTER AND STORE CODES IN TABLES

10

110

RESET REPEAT FLAG THEN OUTPUT CODE

10

140

INTERPRET THE DEVICE NAME OR SCREEN NAME

'DELETE' YES 175

NO

C

C

SCREEN NAME?

YES 145

NO

SET APPROPRIATE OUTPUT DEVICES ON

10

145

DISC FLAG OFF?

YES 199

NO

SCREEN NAME IN TABLE?

YES 199

NO

STORE SCREEN NAME IN TABLE

10
OVERLAYS

**TBLSUB**

E → READ TABLE HEADER CARD

END OF TABLES? YES → RETURN

NO → CORRECT TABLES?

YES → READ IN TABLES

NO → READ TABLES INTO DUMMY ARRAY

E

**ADDSUB**

GLO → INTERPRET FREE FORMAT

CHKSUB

END OF CARD? YES → RETURN

NO → CALL INTEGER CONVERSION ROUTINE

INTSUB

CURSOR POSITION? YES → F

NO → RETURN
OVERLAYS

CMPSUB

DELETETE ALL SCREENS?

YES

SET RECORD POINTERS TO ZERO

NO

RETURN

SEARCH TABLE FOR SCREEN NAME

IS SCREEN NAME FOUND?

YES

DELETE SCREEN NAME AND RECORD POINTERS

NO

RETURN

SCREENS TO BE COMPRESSED

YES

COMPRESS DISC FILES

NO

200

ADJUST POINTER VALUES

LAST POINTER VALUE?

YES

200

NO
SUBROUTINES

I0SUB

STORE VALUE IN OUTPUT BUFFER

IS BUFFER FULL?

YES → CALL WRITE ROUTINE

NO → IS REPEAT FLAG SET?

YES → STORE VALUE IN REPEAT ARRAY

NO → RETURN

INTSUB

CALCULATE STARTING POINT FOR INTEGER ARRAY

MASK BUFFER AND FILL ARRAY

CALL INTEGER CONVERSION LIBRARY SUBROUTINE

RETURN
PROGRAM SBLD ASSISTS THE PROGRAMMER IN BUILDING CRT SCREENS
BY INTERPRETING USER-SUPPLIED CODES FOR CRT CURSOR POSI-
TIONING AND ATTRIBUTES, AND THEN OUTPUTING THE HEX EQUIVA-
LENT CODES, ALONG WITH TEXT STRINGS, TO THE FOLLOWING
DEVICES OR ANY COMBINATION OF THESE DEVICES: CRT, PRINTER,
DISC FILE (TO BE CALLED UP LATER FROM A USER PROGRAM)

PROGRAM SBLD
IMPLICIT INTEGER (A-Z)
DECLARE AND INITIALIZE VARIABLES
REAL CODE, ACODE(50), SNAME, SCRNAM(100)
INTEGER IRPTC(100), TXT(80), COL(2), IBUF(4), LOC(31, 2), IOUT(40)
ICODE(50), MONITR(100, 3), IOV1(4), IOV2(4), IOV3(4), IOV4(4), IOV5(4)
IOV6(4), IOV7(4), ITEXT(50)
LOGICAL JFLAG, JFLG1, JFLG2, JFLG3, CFLAG, BFLAG
COMMON IRPT, IOUT, IOQIT, FLG3, JFLAG, NO
COMMON/BLK1/IERR
COMMON/BLK2/TXT
COMMON/BLK3/IBUF
COMMON/BLK4/CO
COMMON/BLK5/IDISC, LREC, IVAR
GLOBAL COMMON ASSIGNMENTS
COMMON/3GLOB/1BG(768)
EQUIVALENCE (ISG, SCRNAM), (ISG(201), MONITR(1, 1)), (CODE, CO)
DEFINE FILE STATEMENT.
DEFINE FILE 3O(50, 40, U, IVAR)
DATA :CODE, NLOC, MODEL/O, O, O/
ASSIGN OVERLAY NAMES
DATA IOV1/*ER*/ PS/*UB*/, O/
DATA IOV2/*DS*/ CS/*UB*/, O/
DATA IOV3/*CD*/ FS/*UB*/, O/
DATA IOV4/*PA*/ KS/*UB*/, O/
DATA IOV5/*AD*/ DS/*UB*/, O/
DATA IOV6/*TB*/ LS/*UB*/, O/
DATA IOV7/*CM*/ PS/*UB*/, O/
THE NEXT THREE LINES ARE REQUIRED BY THE SYSTEM

C

INLINE
CALLMM,1
BRX: +1
FINI

C SET SIZE OF OUTPUT BUFFER
IOIT=40

C SET DISC, TYP. AND CARD READER
IDISC=30
ITYP=6
IUNIT=5
MNTR=100

C 'LREC' IS THE RECORD LENGTH OF THE DISC FILE
LREC=50
ICRT=0
INSLT=0
IERR=0
IVAR=0
IDUM=4Z7F7F
BFLAG=.FALSE.

C SET FLAGS FOR OUTPUT TO - OFF
FLAG1=.FALSE.
FLAG2=.FALSE.
FLAG3=.FALSE.

C ******************************************************
C READ FIRST CARD TO DETERMINE CORRECT INPUT TABLES BASED
C ON THE MODEL, THE CRT DEVICE NUMBER MAY ALSO BE ASSIGNED
C HERE
READ(IUNIT,900) MODEL,(TXT(I),I=4,30)
CFLAG=.TRUE.
CALL OVERL(IOV5,11)
CALL OVRUN(IOV5,LOC,MLOC,ICRT,I,CFLAG)
C CHECK FOR ERROR IN OVERLAY OR SUBROUTINE
IF(IERR.NE.0) GO TO 199
CFLAG=.FALSE.
CALL OVERLAY TO FIND AND STORE CORRECT TABLES
CALL OVERL(IOV6,11)
CALL OVRJH(IOV6,MODEL,NLOC,NCODE,LOC,ACODE,ICODE,UNIT,ITIME)
CHECK FOR ERROR IN OVERLAY OR SUBROUTINE
IF(IERR.NE.0) GO TO 199
GO TO 5
CONTINUE
CALL OVERLAY TO MONITOR DISCO FILE
CALL OVERL(IOV2,11)
CALL OVRUN(IOV2,II,MNTR,ITYP)
CHECK FOR ERRORS IN OVERLAY OR SUBROUTINE
IF(IERR.NE.0) GO TO 199
SET RECORD NUMBER, SCREEN NUMBER
IVAR=MONITR(II,2)
MSCRN=MONITR(II,1)
CONTINUE
IO=0
SET JFLAG — ON. IF REPEAT CARD ENCOUNTERED SET JFLAG — OFF
JFLAG=.TRUE.
INITIALIZE OUTPUT BUFFER TO BLANKS
SNAME=’
CONTINUE
SET FLAG FOR LOWER CASE LETTERS — OFF
IFLAG=.FALSE.
INITIALIZE CODE APRAY
CO(1)=0
CO(2)=0
READ INPUT FROM CARDS
READ(UNIT,999,END=200) TXT
WRITE(ITYP,2000) TXT
106 C ******************************************************************************
107 C   CHECK FIRST CHARACTER OF INPUT
108 C   IF(TXT(1).EQ.'/') Go TO 100
109 C   IF(TXT(1).EQ.'C'.AND.TXT(2).EQ.' ') Go TO 10
110 C   CHECK FOR COMMAND TO BUILD SCREEN BEFORE STARTING BUILD
111 C   PROCEDURE
112 C   IF(.NOT.BFLAG) Go TO 195
113 C   IF(TXT(1).EQ.'$') Go TO 45
114 C   IF(TXT(1).NE.'(') Go TO 50
115 C  ******************************************************************************
116 C   PROCESS TEXT STRING
117 C   SET FLAG FOR LOWER CASE LETTERS - ON
118 C   IFLAG=.TRUE.
119 C   CALL PACKING OVERLAY FOR TEXT STRING
120  45 CALL OVERL(IOV4,11)
121 C   CALL OVRUN(IOV4,IFLAG)
122 C   CALL FOR ERROR IN OVERLAY OR SUBROUTINE
123 C   IF(IERR.NE.0) Go TO 199
124 C   FINISHED WITH INPUT CARD, SO READ NEXT INPUT CARD
125 C   GO TO 10
126 C  ******************************************************************************
127 C   CALL OVERLAY FOR CODE INTERPRETATION
128  50 CONTINUE
129 C   CALL OVERL(IOV3,11)
130 C   CALL OVRUN(IOV3,ACODE,I,NCODE,ICODE,CFLAG,TIME)
131 C   CALL FOR ERROR IN OVERLAY OR SUBROUTINE
132 C   IF(IERR.NE.0) Go TO 199
133 C   GO TO 10
134 C  100 CONTINUE
135 C ******************************************************************************
136 C   CHECK SECOND CHARACTER OF INPUT
137 C   IF(TXT(2).EQ.'E'.OR.TXT(2).EQ.'B') Go TO 200
138 C   IF(TXT(2).EQ.'D') Go TO 140
C CHECK FOR COMMAND TO BUILD SCREEN BEFORE STARTING BUILD

C PROCEDURE

IF(.NOT.BFLAG) GO TO 195
IF(TXT(2).EQ.'F') GO TO 110
IF(TXT(2).EQ.'C'.OR.TXT(2).EQ.'S') GO TO 140

C **********************************************************************
C CALL OVERLAY FOR CURSOR POSITIONING AND INTEGER INTERPRETATION
C
IRSLT=0
CALL OVERL(IOV5,11)
CALL OVRUN(IOV5,LOC,LOC,IRSLT,I,CFLAG)

C CHECK FOR ERROR IN OVERLAY OR SUBROUTINE
IF(IEMP.NE.0) GO TO 199

C **********************************************************************
C IF REPEAT CARD ENCOUNTERED SET REPEAT FLAG - ON, CALCULATE
C NUMBER OF REPEATS 'IFIN', AND INITIALIZE REPEAT BUFFER
C SUBSCRIPT 'NO'
IF(TXT(2).EQ.'R') GO TO 102
C CHECK FOR COMMAND TO MODIFY CODE TABLES
IF(TXT(2).EQ.'E') GO TO 105

100 GO TO 10

102 IFIN=IRSLT-1
IF(IFIN.LE.0) IFIN=1
JFLAG=.FALSE.
NO=0
GO TO 10

C **********************************************************************
C FOR MODIFYING CODE TABLES SET END OF TABLE TO 'ISTART'
C CALCULATE THE NEW TABLE END 'NCODE', CHECK FOR INPUT ERROR,
C AND READ ADDITIONAL CODES

105 ISTART=NCODE+1
NCODE=NCODE+IRSLT
IF(ISTART.LE.NCODE) GO TO 108
IERR=10
GO TO 199

108 READ(IUNIT,980)(ACODE(J),ICODE(J),ITIME(J),J=ISTART,NCODE)
GO TO 10
**END OF REPEAT INPUT CARDS SO SET Jflag TO - ON**

```c
Jflag=.true.
```

**OUTPUT REPEATED CODE**

```c
do 130 j=1,ifin!
do 120 jj=1,no
```

**CALL IOSUB(IrpT(JJ))**

**CHECK FOR ERROR IN OVERLAY OR SUBROUTINE**

```c
if(ierp.ne.0) go to 199
```

**CONTINUE**

```c
continue
```

**GO TO 10**

**DEVICE CONTROL CARD ENCOUNTERED, SET 'CFLAG' - ON IN ORDER TO USE 'CDESUB' TO INTERPRET DEVICE TYPE OR SCREEN NAME**

```c
cflag=.true.
call overl(iov3,11)
call ovrun(iov3,acode,i,nCode,icode,cflag,itime)
```

**IF(IERR.NE.0) GO TO 199**

```c
if(ierr.ne.0) go to 199
```

**CFLAG=.FALSE.**

```c
if(tat(2).eq.'D') go to 175
if(tat(2).eq.'S') go to 145
```

**CHECK FOR DEVICE TYPE AND SET CORRESPONDING FLAG - ON**

```c
if(code.eq.'typ ') flag1=.true.
if(code.eq.'ct ') flag2=.true.
if(code.eq.'dsc ') flag3=.true.
```

**GO TO 10**

**AN ERROR OCCURS IF SCREEN NAME CARD IS ENCOUNTERED AND DISC FLAG, 'FLAG3', IS NOT SET ON**

```c
if(flag3) go to 150
ierp=3
```

**GO TO 199**

**SET SCREEN NAME TO CODE**

```c
snam=code
```

88
C CHECK FOR DUPLICATE SCREEN NAME. IF DUPLICATE IS FOUND
212 DO 155 JJ=1,100
213 IF(NAME(JJ,1).EQ. ') GO TO 160
214 IF(NAME(JJ,NI).NE.SNAME) GO TO 155
215 IERR=2
216 GO TO 199
217 155 CONTINUE
218 C STORE SCREEN NAME IN TABLE
219 160 IF(NAME(II,1).EQ. ') GO TO 165
220 IERR=14
221 GO TO 199
222 165 NAME(II)=SNAME
223 GO TO 10
224 C ***********************************************************
225 C CALL OVERLAY TO DELETE SCREEN FROM DISC FILE
226 175 IF(.NOT.BFLAG) GO TO 180
227 IERR=16
228 GO TO 199
229 180 NAME=CODE
230 CALL OVERL(IOV7,II)
231 CALL OVRUN(IOV7,SNAME,ITYP,MINTR)
232 C CHECK FOR ERROR IN OVERLAY OR SUBROUTINE
233 IF(IERR.NE.0) GO TO 199
234 GO TO 10
235 195 IERR=14
236 C ***********************************************************
237 C AN ERROR WAS FOUND. CALL ERROR MESSAGE OVERLAY
238 199 CALL OVERL(IOVI,II)
239 CALL OVRUN(IOVI,ITYP)
240 IERR=0
241 FLAG1=.FALSE.
242 FLAG2=.FALSE.
243 FLAG3=.FALSE.
244 GO TO 10
245 C      ***********************************************
246 C      IF NEW SCREEN CARD OR END OF INPUT CARD IS ENCOUNTERED,
247 C      THEN CLEAR ALL BUFFERS AND WRITE TO APPROPRIATE OUTPUT
248 C      DEVICES
249 200  CONTINUE
250 C      IF BUILD FLAG IS NOT SET THEN SET IT ON
251 IF(BFLAG) GO TO 203
252 IF(TXT(2).EQ.'E') GO TO 260
253 BFLAG=.TRUE.
254 GO TO 255
255 C      CHECK FOR REPEAT CARD ERROR
256 203 IF(JFLAG) GO TO 205
257 IERR=11
258 CALL OVERL(IOV1,II)
259 CALL OVRUN(IOV1,ITYP)
260 IERR=0.
261 FLAG3=.FALSE.
262 GO TO 255
263 C      CALL WRITE ROUTINE TO WRITE OUT LAST RECORD
264 205 IF(IO.EQ.0) GO TO 222
265 NEXT=IO+1
266 GO 210 J=NEXT,IQT
267 CALL INSU3(IDUM)
268 210 CONTINUE
269 220 IF(IERR.EQ.0) GO TO 222
270 CALL OVERL(IOV1,II)
271 CALL OVRUN(IOV1,ITYP)
272 FLAG3=.FALSE.
273 IERR=0.
274 GO TO 255
275 C      SET LAST RECORD IN GLOBAL COMMON
276 222 IF(MONITR(II,2).EQ.IVAR) IVAR=IVAR+1
277 MONITR(II,3)=IVAR-1
278 225 M2=MONITR(II,2)
279 M3=MONITR(II,3)
DO 250 J=M2,M3
251 VAR=J
252 READ(IDISC*IVAR) IOUT
253 C WRITE TO OUTPUT DEVICE IF FLAGS ARE SET ON
254 C IF(FLAG1) WRITE(ITYP,910) IOUT
255 C IDUM IS USED WHEN WRITING TO THE CRT TO AVOID UNWANTED
256 C CARRIAGE CONTROL COMMANDS SUCH AS BLANKS, WHICH MIGHT
257 C APPEAR IN THE FIRST POSITION OF 'IOUT'
258 C IF(FLAG2) WRITE(ICRT,920) IDUM,IOUT
259 250 CONTINUE
260 C IF DISC FLAG IS NOT SET ON THEN DELETE SCREEN FROM TABLE
261 255 IF(FLAG3) GO TO 260
262 SCRNAM(I)=''
263 MONITR(I,1)=0
264 MONITR(I,2)=0
265 MONITR(I,3)=0
266 260 CONTINUE
267 IF(TXT(2).EQ.'B') GO TO 1
268 WRITE(ITYP,4000)
269 900 FORMAT(I3,77A1)
270 910 FORMAT(18A4,40A2)
271 920 FORMAT(41A2)
272 980 FORMAT(A4,Z4,I4)
273 999 FORMAT(80A1)
274 2000 FORMAT(1X,'FILE READ',2X,80A1)
275 4000 FORMAT(1X,'END OF PROGRAM')
276 STOP
277 END
278 C
C ERRSUB WRITES ERROR MESSAGES DEPENDENT ON THE ERROR NUMBER
C STORED IN 'IERR'
C
C OVERLAY ERRSUB(IYYPE)
C COMMON/BLK1/IERR
C GO TO (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17),IERR
C
C 1 WRITE(IYYPE,51)
C RETURN
C 2 WRITE(IYYPE,52)
C RETURN
C 3 WRITE(IYYPE,53)
C RETURN
C 4 WRITE(IYYPE,54)
C RETURN
C 5 WRITE(IYYPE,55)
C RETURN
C 6 WRITE(IYYPE,56)
C RETURN
C 7 WRITE(IYYPE,57)
C RETURN
C 8 WRITE(IYYPE,58)
C RETURN
C 9 WRITE(IYYPE,59)
C RETURN
C 10 WRITE(IYYPE,60)
C RETURN
C 11 WRITE(IYYPE,61)
C RETURN
C 12 WRITE(IYYPE,62)
C RETURN
C 13 WRITE(IYYPE,63)
C RETURN
C 14 WRITE(IYYPE,64)
C RETURN
344 15 WRITE(IOUTPUT, 65)
345 RETURN
346 16 WRITE(IOUTPUT, 66)
347 RETURN
348 17 WRITE(IOUTPUT, 67)
349 RETURN
350 51 FORMAT('Disc is full')
351 52 FORMAT('Screen name is not unique')
352 53 FORMAT('(/s) card found without a preceding (/c...dsc) card')
353 54 FORMAT('Maximum number of screens exceeded')
354 55 FORMAT('Correct table not found')
355 56 FORMAT('End of text string not found')
356 57 FORMAT('Repeat limit exceeded')
357 58 FORMAT('Code not found')
358 59 FORMAT('Illegal input value')
359 60 FORMAT('No numeric value found on (/s) card')
360 61 FORMAT('A (/f) card was never found following a (/p) card')
361 62 FORMAT('Code table missing')
362 63 FORMAT('Disc is set on without a screen name')
363 64 FORMAT('Missing (/b) card')
364 65 FORMAT('Name of screen to be deleted was not found')
365 66 FORMAT('(/d) card encountered during a build')
366 67 FORMAT('Invalid cursor position')
367 RETURN
368 END
DCSUSR MONITORS THE DISC OUTPUT AND KEEPS TRACK OF THE NUMBER OF RECORDS USED BY A SCREEN.

OVERLAY DCSUSR(IIF,MNTR,ITYP)

"MONIT" IS A 100X3 TABLE WHERE COLUMN 1 IS THE SCREEN NUMBER, COLUMN 2 IS THE FIRST RECORD OF A SCREEN, AND COLUMN 3 IS THE LAST RECORD OF A SCREEN.

INTEGER MNTR(100,3)
REAL SCRNAM(100)
COMMON/BLKI/IERR
COMMON/SGLOB/IBG(768)

PUT TABLE IN GLOBAL COMMON
EQUIVALENCE (IBG,SCRNAM),(IBG(201),MNTR(1,1))
FIND THE LOCATION OF THE FIRST AVAILABLE RECORD ON DISC

DO 100 J=1,MNTR
II=J
IF(MNTR(J,1).EQ.0) GO TO 120
CONTINUE
ALL RECORDS ARE BEING USED, SO RETURN WITH ERROR CODE
IERR=4
RETURN

IF(II.EQ.1) GO TO 130
NN=II-1
SCREEN NUMBER IS LAST SCREEN NUMBER PLUS 1
MNTR(II,1)=MNTR(NN,1)+1
FIRST RECORD OF THE SCREEN IS THE LAST RECORD USED PLUS 1
MNTR(II,2)=MNTR(NN,3)+1
RETURN
FIRST RECORD IS AVAILABLE

MONTR(1,1)=1
MONTR(1,2)=1
RETURN
END
**PAKSUB PACKS TEXT STRINGS AND CALLS 'IOUSR' FOR OUTPUT**

```fortran
OVERLAY PAKSUB(IFLAG)
INTEGER TXT(30)
LOGICAL IFLAG
COMMON/BLK1/IERR.
COMMON/BLK2/TXT
MASK=4ZFF00
MASK2=4Z007F
I=0
N=0
DO 260 J=1,79
   IF(TXT(J).NE.'$'.AND.TXT(J+1).NE.'(') GO TO 225
   END OF TEXT STRING IS AT M-1
   M=J
   TXT(J+1)=
   GET RID OF DELIMITERS
225 TXT(J)=TXT(J+1)
250 CONTINUE
   IF(M.NE.0) GO TO 265
260 CONTINUE
   IERR=6
   RETURN
265 DO 270 J=2,M,2
   I=I+1
   TXT(J)=ISHFT(TXT(J),-8)
```

**Note:** The code snippet appears to be a part of a FORTRAN program intended to process text strings, possibly for output formatting or error checking. The comments indicate it packs text strings and calls a subroutine named 'IOUSR' for output. The code includes logic for handling delimiters, skipping non-alphabetic characters, and performing bit operations on the text strings.
PACK TEXT STRING

TXT(I) = IOR(TXT(J-1), TXT(J))

CHECK FOR A BLANK IN THE LAST BYTE AND CHANGE IT TO A DELETE

IF(J.EQ.M) TXT(I) = IOR(TXT(I), MASK2)

CALL OUTPUT BUFFER SUBROUTINE 'IOSUB'

CALL IOSUB(TXT(I))

IF(IERR .NE. 0) RETURN}

CONTINUE

RETURN

END
C ******************************************************************************************************************
C TELSUB FINDS THE CORRECT TABLE AND READS IT INTO STORAGE
C ******************************************************************************************************************
C OVERLAY TELSUB(MODEL,INLOC,ICODE,LOC,ACODE,ICODE,IUNIT,ITIME)
C 'ACODE' IS THE CODE TABLE
C REAL ACODE(50)
C 'LOC' IS THE CURSOR ADDRESS TABLE
C 'ICODE' IS THE HEX PART OF THE CODE TABLE
C INTEGER LOC(81,2),ICODE(50),ITIME(50)
C COMMON/IHLK1/IERR
C N=1
C SET TYPE OF TABLE TO CURSOR ADDRESS
C TSPEC='ADD'
C CONTINUE
C READ TABLE HEADER CARD
C READ(IUNIT,350) TKIND,NTYPE,NUM
C CHECK FOR THE END OF THE TABLES
C IF(TKIND.EQ.1/*'*/) GO TO 310
C IF(N.EQ.1) IERR=5
C RETURN
C SET LENGTH OF TABLE
C INLOC=NUM
C CHECK FOR CORRECT TABLE
C IF(NTYPE.EQ.MOODEL.AND.TKIND.EQ.TSPEC) GO TO 325
C IGNORE INCORRECT TABLE
C DO 320 J=1,NUM
C READ(IUNIT,355)
C CONTINUE
C GO TO 305
C CONTINUE
C*********************************************************************** READ CURSOR ADDRESS TABLE
C READ(IUNIT,355)(LOC(J,JJ),JJ=1,2),J=1,NLOC
C SET TYPE OF TABLE TO CODE
C TSPEC='CODE'
485  C     READ TABLE HEADER CARD
486  READ(IUNIT,350) TKIND,NTYPE,YCODE
487  C     CHECK FOR CORRECT TABLE
488  IF(NTYPE.NE.CODEL.OR.TKIND.NE.TSPEC) GO TO 345
489  C     READ CODE TABLE
490  READ(IUNIT,370)(ACODE(J),ICODE(J),ITIME(J),J=1,NCODE)
491  N=N+1
492  C     CLEAR OUT REST OF TABLES
493  GO TO 305
494  345  IERR=12
495  350  FORMAT(A4,I4,I3)
496  355  FORMAT(IY)
497  355  FORMAT(I3,A2)
498  370  FORMAT(A4,Z4,I4)
499  RETURN
500  END
**ADDSSUR SEARCHES CURSOR ADDRESS TABLE FOR CURSOR POSITIONING CODE**

**OVERLAY ADDSSUR(LOC,ITOL,IRSLT,I,CFLAG)**

**INTEGER LOC(81,2),TXT(80)**

**LOGICAL CFLAG**

**COMMON/BLKI/IERR**

**COMMON/BLK2/TXT**

**KOUNT=1**

**ISTAT=0**

**N=3**

IF 'CFLAG' IS SET ON THEN 'ADDSSUR' IS USED ONLY TO AID IN INTEGER CONVERSION AND TABLES ARE NOT SEARCHED

**IF(CFLAG) N=4**

**I=0**

**SET CURSOR POSITIONING COMMAND**

**LCA=4Z13F6**

**CALL SUBROUTINE TO PICK UP NON-BLANK VALUES**

**CALL CHKSUB(N,I)**

**IF(IERR,NE.0) RETURN**

**CHECK FOR END OF CARD**

**IF(I.EQ.0) RETURN**

**CALL INTEGER CONVERSION ROUTINE**

**CALL INTSUB(I,IRSLT,ISTAT)**

**IF(IERR,NE.0) RETURN**

**IF(TXT(2),NE.'M') GO TO 660**

**SEARCH CURSOR ADDRESS TABLES USING A BINARY SEARCH**

**LOW=1**

**I HIGH=ITOL**

**IF(KOUNT,NE.1) GO TO 620**

**IF(IRSLT.GT.24) IERR=17**

'ADDSSUR' IS BEING USED TO POSITION CURSOR SO OUTPUT CURSOR
C POSITIONING COMMAND
CALL I0SUB(LCA)
IF(IERP.NE.0) RETURN
IF(IHIGH.LT.LOW) GO TO 640
IMID=(LOW+IHIGH)/2
IF(IRSIT.GT.LOC(IMID,1)) LOW=IMID+1
IF(IRSIT.LT.LOC(IMID,1)) IHIGH=IMID-1
IF(IRSIT.EQ.LOC(IMID,1)) GO TO 650
GO TO 620
C ADDRESS NOT FOUND, ERROR CODE IS RETURNED
IERR=8
RETURN
C OUTPUT RESULT
CALL I0SUB(LOC(IMID,2))
KOUNT=KOUNT+1
IF(IERP.NE.0) RETURN
CONTINUE
IF(I.NE.0) GO TO 610
RETURN
END
LJJ
llll
C
>&
L'L
..,
(.)
-b-4
CI
**C**

553 C ************************************************************
554 C CDESUB SEARCHES THE CODE TABLES AND OUTPUTS THE HEX EQUIVALENT
555 C ************************************************************
556 C OVERLAY CDESUB(ACODE, I,ICODE, ICODE, CFLAG, ITIME)
557 C REAL CODE,ACODE(50)
558 C INTEGER IRBUF(4),C0(2),ICODE(50),ITIME(50)
559 C LOGICAL CFLAG
560 C COMMON/BLK1/IERR
561 C COMMON/BLK3/IRBUF
562 C COMMON/BLK4/C0'
563 C 'CODE' WILL CONTAIN THE PACKED 'C0' VALUES
564 C EQUIVALENCE (CODE,C0)
565 N=1
566 I=0
567 IDEL=47F7F
568 IF(CFLAG) N=3
569 C CALL SUBROUTINE TO PICK UP NON-BLANK VALUES
570 700 CALL CHKSUB(N,I)
571 IF(IERR.NE.0) RETURN
572 IF(I.EQ.2) IRBUF(3)=42000
573 C CHECK FOR END OF CARD
574 IF(I.EQ.0) RETURN
575 C SHIFT CHARACTERS TO THE RIGHT 8 BITS
576 DO 725 J=2,I,2
577 IRBUF(J)=ISHFT(IRBUF(J),-8)
578 725 CONTINUE
579 C PACK 'IRBUF' ARRAY INTO 'C0' ARRAY
580 C0(1)=IOR(IRBUF(1),IRBUF(2))
581 C0(2)=IOR(IRBUF(3),IRBUF(4))
582 C 'CODE' NOW CONTAINS 'C0(1)' AND 'C0(2)' PACKED INTO FOUR
583 C BYTES
584 IF(CFLAG) RETURN
585 C    SEARCH TABLE OF CODE VALUES
586 DO 750 J=1,NCODE
587       JJ=J
588 IF(CODE.EQ.ACODE(J)) GO TO 750
589 CONTINUE
590 C    CODE NOT FOUND, ERROR CODE RETURNED
591 IERR=8
592 RETURN
593 C    OUTPUT HEX EQUIVALENT
594 CALL IOSUB(ICODE(JJ))
595 IF(ITIME(JJ).EQ.0) GO TO 770
596 NUM=ITIME(JJ)
597 DO 780 J=1,NUM
598 CALL IOSUB(IDEL)
599 CONTINUE
600 770 CONTINUE
601 IF(IERR.NE.0) RETURN
602 IF(I1.NE.0) GO TO 700
603 RETURN
604 END
C C ************************************************************
C  CMPSUR DELETES SCREENS FROM THE DISC FILE AND THEN
C  THE FILE AND ADJUSTS THE DISC MONITOR TABLE ACCORDINGLY
C ************************************************************
C OVERLAY CMPSUR(SNAME,ITYP,MNTR)
C DECLARE AND INITIALIZE VARIABLES
C REAL SCRNAM(100)
C INTEGER MONITR(100,3),ITXT(40)
C COMMON/BLK1/IERR
C COMMON/BLK5/IDISC,LREC,IVAR
C COMMON/3GLOB/IBG(768)
C EQUIVALENCE *(ISG,SCRNAM),(ISG(201),MONITR(1,1))
C N0TUSE=0
C MNTR2=MNTR-1
C WRITE(ITYP,901) SNAME
C CHECK FOR A 'DELETE ALL' COMMAND
C IF(SNAME.NE.'/* ') GO TO 300
C JJ=1
C MOVE BLANKS INTO SCREEN NAME ARRAY AND ZEROS INTO DISC
C MONITOR TABLE
C DO 150 J=JJ,MNTR
C SCRNAM(J)='
C MONITR(J,1)=0
C MONITR(J,2)=0
C MONITR(J,3)=0
C CONTINUE
C IF(JJ.EQ.1) WRITE(ITYP,920)
C GO TO 900
C CHECK FOR SCREEN NAME IN SCREEN NAME ARRAY
C DO 350 J=1,MNTR
C N=J
C IF(SCRNAM(J).EQ.SNAME) GO TO 400
C CONTINUE
C IERR=15
C RETURN
C GO TO 900
C DELETE SCREEN NAME FROM SCREEN NAME ARRAY
400 DO 450 J=N,MNTR2
410 II=J+1
420 SCRNAM(J)=SCRNAM(II)
430 C CHECK FOR LAST SCREEN NAME IN ARRAY
440 IF(SCRNAM(II).NE."") GO TO 450
450 C CALCULATE NUMBER OF RECORDS NOT USED
460 NOUTUSE=LREC-MONITR(J,3)
470 GO TO 460
450 CONTINUE
480 C SET LAST SCREEN NAME TO A BLANK
490 SCRNAM(MNTR)="
500 C WRITE ON FIRST RECORD OF DELETED SCREEN
510 IWRT=MONITR(N,2)
520 M=M+1
530 C CHECK FOR EXISTANCE OF SCREEN TO BE MOVED UP
540 IF(MONITR(M,1).EQ.0) GO TO 550
550 C CHECK FOR LAST SCREEN
560 IF(MONITR(N,1).EQ.MNTR) GO TO 650
570 C READ FIRST LINE OF NEXT SCREEN
580 IREAD=MONITR(M,2)
590 C CALCULATE NUMBER OF RECORDS IN USE AFTER SCREEN IS DELETED
600 IUSE=LREC-(IREAD-IWRT)-NOUTUSE
610 C DELETE SCREEN FROM DISC FILE AND MOVE ALL OTHER SCREENS UP
500 DO 550 J=IWRT,IUSE
520 IVAR=IREAD
530 READ(IDISC,IVAR) ITXT
540 IVAR=J
550 WRITE(IDISC,IVAR) ITXT
560 IREAD=IREAD+1
570 CONTINUE
C ADJUST MONITOR TABLE TO CORRESPOND WITH NEW SCREEN POSITION
C ON DISC FILE
550 DO 560 J=M,MNTR
570 JJ=J-1
575 MONITR(JJ,3)=MONITR(JJ,2)+(MONITR(J,3)-MONITR(J,2))
577 MONITR(J,2)=MONITR(JJ,3)+1
578 IF(MONITR(JJ,2).EQ.MONITR(JJ,3)) GO TO 100
560 CONTINUE
565 JJ=MNTR
560 GO TO 100
C WRITE OUT SCREEN NAME ARRAY AND MONITOR FILE
580 DO 590 J=1,10
584 WRITE(IYPE,950) MONITR(J,1),SCRNAM(J),MONITR(J,2),MONITR(J,3)
585 CONTINUE
580 FORMAT(5A4)
582 FORMAT(5A4,'ALL FILES DELETED')
583 FORMAT(5A13,2X,A4,2X,I4,2X,I4)
585 RETURN
590 END
SUBROUTINES

CHKSUB PROCESSES A CARD BY IGNORING BLANKS AND RETURNING THE
NON-BLANK VALUES IN A BUFFER CALLED 'IBUF'

SUEROUTINE CHKSUB(N,I)
INTEGER TXT(60), IBUF(4)
COMMON/BLK1/IEPR
COMMON/BLK2/TXT
COMMON/BLK3/IBUF
ITYP=6

SET MASK VALUES
MASK=4ZFFCO
MASK2=420020
I=0

MASK 'IBUF'
DO 405 J=1,4
IBUF(J)=MASK2
CONTINUE

CHECK FOR A NON-BLANK VALUE
DO 410 J=N,80
NN=J
IF(TXT(J).NE.' ') GO TO 420
CONTINUE
RETURN

FILL BUFFER 'IBUF' AND CHECK FOR NEXT BLANK
DO 430 J=NN,80
N=J
IF(TXT(J).EQ.' ') RETURN
TXT(J)=IAND(TXT(J),MASK)
I=I+1
IF(I.LE.4) GO TO 425
IBUF DIMENSION'S EXCEEDED

IRE=9
RETURN

IBUF(I)=TXT(J)
CONTINUE

SET LAST BYTE TO BLANK TO AVOID AN INFINITE LOOP

TXT(J)="'
RETURN
END

********************************************************************************************************
INTSUB CONVERTS ASCII TO INTEGER
********************************************************************************************************
SUBROUTINE INTSUB(I,IRSLT,ISTAT)
INTEGER IBUF(4),INTG(8)
COMMON/BLK1/IRE
COMMON/BLK3/IBUF
ITYP=6

INITIALIZE INTEGER ARRAY
DO 500 J=1,3
INTG(J)=4Z3020
CONTINUE
K=1

CALCULATE STARTING POINT
M=I-1
DO 510 J=4,3
MASK 'IBUF'
INTG(J)=IOR(IBUF(K),4Z0020)

CHECK FOR ILLEGAL ASCII VALUE
IF(INTG(J).LT.4Z3020.OR.INTG(J).GT.4Z3920) IERR=9
K=K+1
CONTINUE

CALL CONVERSION ROUTINE 'IABIN' AND STORE INTEGER IN 'IRSLT'
IIRSLT=IABIN(INTG,ISTAT)
RETURN
END
I9SUB STORES OUTPUT VALUES IN THE BUFFER ARRAY 'IOUT'. WHEN 'IOUT' IS FULL THE OUTPUT SUBROUTINE 'WRITSUB' IS CALLED

SUBROUTINE I9SUB(IDUM)
INTEGER IRPT(100),IOUT(40)
LOGICAL FLAG3,JFLAG
COMMON IRPT,IOUT,IO,IOIT,FLAG3,JFLAG,NO
COMMON /SLK1/ IERR

ITYP=6

INCREMENT THE ARRAY SUBSCRIPT 'IO' BY 1
I0=I0+1

STORE OUTPUT IN OUTPUT BUFFER ARRAY 'IOUT'
IOUT(10)=IDUM

WHEN BUFFER IS FULL CALL THE OUTPUT SUBROUTINE 'WRITSUB'
IF(IO.EQ.IOIT) CALL WRITSUB(FLAG3,IOUT,IO)
IF(IERR.NE.0) RETURN
IF(JFLAG) RETURN

'JFLAG' IS NOT SET SO A REPETITION IS REQUIRED
NO=NO+1

CHECK LENGTH OF REPETITION ARRAY
IF(NO.LE.100) GO TO 810
IEPP=7
RETURN

STORE VALUES TO BE REPEATED IN 'IRPT' ARRAY

810 IRPT(NO)=IDUM
RETURN
END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
WRITSUB OUTPUTS THE ARRAY 'IOUT' TO DISC
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
SUBROUTINE WRITSUB(FLAG3,IOUT,IO)
INTEGER IOUT(40)
LOGICAL FLAG3
COMMON/BLK1/IERR
COMMON/BLK5/IDISC,LREC,IVAR
ITYP=6
C CHECK FOR AVAILABLE DISC SPACE
IF(IVAR.LE.LREC) GO TO B10
IERR=1
RETURN
IF(FLAG3.AND.IVAR.GT.0) GO TO 825
IERR=13
RETURN
WRITE(IDISC,IVAR) IOUT
C RE-INITIALIZE OUTPUT ARRAY 'IOUT'
DO 850 J=1,40
IOUT(J)=422020
CONTINUE
I0=0
RETURN
END
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
REAL SCRNAM(100)
INTEGER MONITP(100,3),ITXT(40)
COMMON/RGLOB/IBG(768)
EQUIVALENCE (IBG,SCRNAM),(IBG(201),MONITP(1,1))
DEFINE FILE 30(50,40,U,IVAR)
IDISC=30
ITYP=2
IPF=4Z1R5B
ICUH=4Z1R4B
ICSC=4Z1B4A
IDUM=4Z7F7F
DO 100 J=1,100
II=J
IF(SCRNAM(J).EQ.SNAME) GO TO 200
100 CONTINUE
WRITE(ICRT,500)
RETURN
200 ISTART=MONITP(I,2)
IFIN=MONITP(I,3)
IF(ICLEAR.NE.1) GO TO 225
WRITE(ICRT,525) IPF,ICUH,ICSC
CALL WAIT(20,0,1HUD)
DO 250 J=ISTART,IFIN
   IVAR=J
   READ(IDISC,IVAR) ITXT
   WRITE(ICRT,550) IDUN,ITXT
250 CONTINUE
550 FORMAT(’ ’,’SCREEN NAME NOT FOUND’)
525 FORMAT(3A2)
550 FORMAT(41A2)
RETURN
END
<table>
<thead>
<tr>
<th>CODE</th>
<th>DESCRIPTION</th>
<th>CODE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Cursor/Pointer Up</td>
<td>VNM</td>
<td>Video Normal</td>
</tr>
<tr>
<td>CPD</td>
<td>Cursor/Pointer Down</td>
<td>LCF</td>
<td>Local Copy Off</td>
</tr>
<tr>
<td>CPR</td>
<td>Cursor/Pointer Right</td>
<td>LCO</td>
<td>Local Copy On</td>
</tr>
<tr>
<td>CPL</td>
<td>Cursor/Pointer Left</td>
<td>VUL</td>
<td>Video Underline</td>
</tr>
<tr>
<td>CPM</td>
<td>Cursor/Pointer Return</td>
<td>VDM</td>
<td>Video Dim</td>
</tr>
<tr>
<td>CUH</td>
<td>Cursor/Pointer Home</td>
<td>VDR</td>
<td>Dim Reverse</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear Line from Cursor</td>
<td>VBL</td>
<td>Video Blink</td>
</tr>
<tr>
<td>CHD</td>
<td>Character Delete</td>
<td>VBR</td>
<td>Video Blink Reverse</td>
</tr>
<tr>
<td>CHI</td>
<td>Character Insert</td>
<td>PFT</td>
<td>Protect Field – Tab</td>
</tr>
<tr>
<td>CIN</td>
<td>Character Insert On</td>
<td>SPF</td>
<td>Protect Field</td>
</tr>
<tr>
<td>CIF</td>
<td>Character Insert Off</td>
<td>EPF</td>
<td>End Protect Field</td>
</tr>
<tr>
<td>SCU</td>
<td>Scroll Up</td>
<td>KBL</td>
<td>Keyboard Lock</td>
</tr>
<tr>
<td>SCD</td>
<td>Scroll Down</td>
<td>KDU</td>
<td>Keyboard Unlock</td>
</tr>
<tr>
<td>NPG</td>
<td>Next Page</td>
<td>QUO</td>
<td>Quote Mode On</td>
</tr>
<tr>
<td>PPG</td>
<td>Previous Page</td>
<td>BEP</td>
<td>Set Beep Character</td>
</tr>
<tr>
<td>PFN</td>
<td>Protect Format On</td>
<td>STX</td>
<td>Start of Text</td>
</tr>
<tr>
<td>PFF</td>
<td>Protect Format Off</td>
<td>ETX</td>
<td>End of Text</td>
</tr>
<tr>
<td>HTS</td>
<td>Horizontal Tab Set</td>
<td>EPN</td>
<td>End Protect, Non-Alpha</td>
</tr>
<tr>
<td>HTC</td>
<td>Horizontal Tab Clear</td>
<td>UCO</td>
<td>Upper Case Only</td>
</tr>
<tr>
<td>VRV</td>
<td>Highlight on, Video</td>
<td>DEL</td>
<td>Delete</td>
</tr>
<tr>
<td>RCA</td>
<td>Read Cursor Address</td>
<td>VFF</td>
<td>Video Field Off</td>
</tr>
<tr>
<td></td>
<td>Reverse</td>
<td>SPA</td>
<td>Space</td>
</tr>
</tbody>
</table>

**Video Codes**
<table>
<thead>
<tr>
<th>CODE</th>
<th>HEX</th>
<th>TIMING</th>
<th>CODE</th>
<th>HEX</th>
<th>TIMING</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1B41</td>
<td>0</td>
<td>VNM</td>
<td>1B34</td>
<td>1</td>
</tr>
<tr>
<td>CPD</td>
<td>1B42</td>
<td>0</td>
<td>LCF</td>
<td>1B3A</td>
<td>0</td>
</tr>
<tr>
<td>CPR</td>
<td>1B43</td>
<td>0</td>
<td>LCO</td>
<td>1B3B</td>
<td>0</td>
</tr>
<tr>
<td>CPL</td>
<td>1B44</td>
<td>0</td>
<td>VUL</td>
<td>1B61</td>
<td>1</td>
</tr>
<tr>
<td>CPM</td>
<td>1B47</td>
<td>0</td>
<td>VDM</td>
<td>1B63</td>
<td>1</td>
</tr>
<tr>
<td>CUH</td>
<td>1B48</td>
<td>0</td>
<td>VDR</td>
<td>1B64</td>
<td>1</td>
</tr>
<tr>
<td>CLC</td>
<td>1B4B</td>
<td>1</td>
<td>VBL</td>
<td>1B65</td>
<td>1</td>
</tr>
<tr>
<td>CHD</td>
<td>1B50</td>
<td>6</td>
<td>VBR</td>
<td>1B66</td>
<td>1</td>
</tr>
<tr>
<td>CHI</td>
<td>1B40</td>
<td>6</td>
<td>PFT</td>
<td>1B67</td>
<td>1</td>
</tr>
<tr>
<td>CIN</td>
<td>1B51</td>
<td>0</td>
<td>SPF</td>
<td>1B68</td>
<td>1</td>
</tr>
<tr>
<td>CIF</td>
<td>1B52</td>
<td>0</td>
<td>EPF</td>
<td>1B69</td>
<td>1</td>
</tr>
<tr>
<td>SCU</td>
<td>1B53</td>
<td>0</td>
<td>KBL</td>
<td>1B45</td>
<td>0</td>
</tr>
<tr>
<td>SCD</td>
<td>1B54</td>
<td>0</td>
<td>KBU</td>
<td>1B46</td>
<td>0</td>
</tr>
<tr>
<td>NPG</td>
<td>1B55</td>
<td>0</td>
<td>QUO</td>
<td>1B5A</td>
<td>0</td>
</tr>
<tr>
<td>PPG</td>
<td>1B56</td>
<td>0</td>
<td>BEP</td>
<td>1B59</td>
<td>3</td>
</tr>
<tr>
<td>PFN</td>
<td>1B57</td>
<td>1</td>
<td>STX</td>
<td>027F</td>
<td>0</td>
</tr>
<tr>
<td>PFF</td>
<td>1B58</td>
<td>0</td>
<td>ETX</td>
<td>037F</td>
<td>0</td>
</tr>
<tr>
<td>HTS</td>
<td>1B31</td>
<td>0</td>
<td>EPN</td>
<td>1B70</td>
<td>1</td>
</tr>
<tr>
<td>HTC</td>
<td>1B32</td>
<td>0</td>
<td>UCO</td>
<td>1B4E</td>
<td>0</td>
</tr>
<tr>
<td>VRV</td>
<td>1B33</td>
<td>1</td>
<td>DEL</td>
<td>7F7F</td>
<td>0</td>
</tr>
<tr>
<td>RCA</td>
<td>1B7C</td>
<td>2</td>
<td>VFF</td>
<td>1B62</td>
<td>0</td>
</tr>
</tbody>
</table>

**Video Codes**
<table>
<thead>
<tr>
<th>Row and Column Number</th>
<th>Cursor Positioning Sequence</th>
<th>Row and Column Number</th>
<th>Cursor Positioning Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>@@</td>
<td>21</td>
<td>AE</td>
</tr>
<tr>
<td>01</td>
<td>@A</td>
<td>22</td>
<td>AF</td>
</tr>
<tr>
<td>02</td>
<td>@B</td>
<td>23</td>
<td>AG</td>
</tr>
<tr>
<td>03</td>
<td>@C</td>
<td>24</td>
<td>AH</td>
</tr>
<tr>
<td>04</td>
<td>@D</td>
<td>25</td>
<td>AI</td>
</tr>
<tr>
<td>05</td>
<td>@E</td>
<td>26</td>
<td>AJ</td>
</tr>
<tr>
<td>06</td>
<td>@F</td>
<td>27</td>
<td>AK</td>
</tr>
<tr>
<td>07</td>
<td>@G</td>
<td>28</td>
<td>AL</td>
</tr>
<tr>
<td>08</td>
<td>@H</td>
<td>29</td>
<td>AM</td>
</tr>
<tr>
<td>09</td>
<td>@I</td>
<td>30</td>
<td>AN</td>
</tr>
<tr>
<td>10</td>
<td>@J</td>
<td>31</td>
<td>AO</td>
</tr>
<tr>
<td>11</td>
<td>@K</td>
<td>32</td>
<td>B@</td>
</tr>
<tr>
<td>12</td>
<td>@L</td>
<td>33</td>
<td>BA</td>
</tr>
<tr>
<td>13</td>
<td>@M</td>
<td>34</td>
<td>BB</td>
</tr>
<tr>
<td>14</td>
<td>@N</td>
<td>35</td>
<td>BC</td>
</tr>
<tr>
<td>15</td>
<td>@O</td>
<td>36</td>
<td>BD</td>
</tr>
<tr>
<td>16</td>
<td>@Q</td>
<td>37</td>
<td>BE</td>
</tr>
<tr>
<td>17</td>
<td>AA</td>
<td>38</td>
<td>BF</td>
</tr>
<tr>
<td>18</td>
<td>AB</td>
<td>39</td>
<td>BG</td>
</tr>
<tr>
<td>19</td>
<td>AC</td>
<td>40</td>
<td>BH</td>
</tr>
<tr>
<td>20</td>
<td>AD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Cursor Positioning Table**
CRT REQUIREMENTS

The table on the following page shows examples of video codes from four different CRT brands. These brands are: OMRON 8030, ANN ARBOR 531E, ADDS 900 and HEWLETT PACKARD 2640B. It should be noted that codes may exist for one CRT brand and not for another. If a code does not exist for a CRT brand, it is represented by a blank in the table. The OMRON and HEWLETT PACKARD models use HEX code representation and the ANN ARBOR and ADDS models use standard ASCII code representation for their video codes. The standard ASCII code may be converted to HEX representation using the ASCII code table shown below. The main CRT requirements for use with the screen build package are:

1. A new set of tables must be added to the tables section of the package for each different CRT brand.

2. The video codes must be one or two byte(s) HEX codes.

3. For CRT brands whose video codes are only one byte in length (i.e., codes not preceded by an escape byte) a null or delete byte must precede the code.

4. Timing must be considered for each new CRT brand, and the proper number of delete characters must be added to the tables.

<table>
<thead>
<tr>
<th>ASCII CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b )</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
</tr>
<tr>
<td>0111</td>
</tr>
<tr>
<td>1000</td>
</tr>
<tr>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
</tr>
<tr>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
</tr>
<tr>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
</tr>
<tr>
<td>1111</td>
</tr>
<tr>
<td>OMRON 8030 HEX</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>CURSOR UP</td>
</tr>
<tr>
<td>CURSOR DOWN</td>
</tr>
<tr>
<td>CURSOR RIGHT</td>
</tr>
<tr>
<td>CURSOR LEFT</td>
</tr>
<tr>
<td>CURSOR RETURN</td>
</tr>
<tr>
<td>CURSOR HOME</td>
</tr>
<tr>
<td>CLEAR SCREEN</td>
</tr>
<tr>
<td>CLEAR LINE</td>
</tr>
<tr>
<td>LINE INSERT</td>
</tr>
<tr>
<td>LINE DELETE</td>
</tr>
<tr>
<td>CHARACTER</td>
</tr>
<tr>
<td>CHARACTER</td>
</tr>
<tr>
<td>PROTECT</td>
</tr>
<tr>
<td>ESCAPE</td>
</tr>
<tr>
<td>DELETE</td>
</tr>
<tr>
<td>NULL</td>
</tr>
<tr>
<td>BELL</td>
</tr>
<tr>
<td>LOCK KEYBOARD</td>
</tr>
<tr>
<td>UNLOCK</td>
</tr>
<tr>
<td>UNPROTECT</td>
</tr>
</tbody>
</table>

* Must be preceded by an escape character (1B HEX)