Volume I

TOWARD THE DEVELOPMENT OF A UNIVERSAL PROGRAMMING/DOCUMENTATION
SYSTEM FOR PROGRAMMABLE CONTROLLERS ON A HOST MICROCOMPUTER

by

John Andrew Snader

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Industrial Engineering and Operations Research

APPROVED:

M. P. Deisenroth, Chairman

C. E. Nunnally

D. L. Price

October 4, 1985
Blacksburg, Virginia
This thesis presents methodologies for the operation of a programming/documentation system for a Texas Instruments Model 530 programmable controller. The methodologies include: 1) a means to store information about the structure of a ladder diagram and display the ladder diagram on a CRT screen, 2) a means to convert the ladder diagram information into its equivalent Boolean code, and 3) a means to convert Boolean code into its equivalent ladder diagram information.

The methodologies were implemented in a programming/documentation system program capable of running on a host IBM personal computer. The system program allows an operator to create and display ladder diagram programs, save the programs on a diskette in either ladder diagram or Boolean code format, and retrieve the programs from a diskette.

The thesis also presents a review of selected programmable controller instruction sets. Based on the review and knowledge gained from the development of the system program, conclusions are drawn concerning the
feasibility of creating a universal programming/documentation system. In addition, a general strategy for creating the universal system is outlined.
ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my chairman, Dr. Michael P. Deisenroth, for his endless patience and encouragement and for always managing to convince me that there was never anything to worry about. Additionally, were it not for his work on the primary concepts behind the ladder to Boolean conversion algorithm, its development would have been a long and tedious process.

I owe much thanks also to for her relentless support through all the rough times and for helping me believe that I could meet the challenges.
# TABLE OF CONTENTS

1.0 CHAPTER 1. INTRODUCTION ............................................. 1
   1.1 Problem Statement .................................................. 3
   1.2 Objective .......................................................... 4
       1.2.1 Phase One Tasks ............................................ 6
       1.2.2 Phase Two Tasks ............................................ 8
   1.3 Terminology ......................................................... 9

2.0 CHAPTER 2. LITERATURE REVIEW ...................................... 13
   2.1 The History of Programmable Controllers ......................... 13
       2.1.1 Hardware Developments .................................... 14
       2.1.2 Software Developments .................................... 16
   2.2 The Future of Programmable Controllers ......................... 17
   2.3 Universal Programming/Documentation Systems .................... 17

3.0 CHAPTER 3. METHODOLOGY .............................................. 20
   3.1 System Overview .................................................. 20
   3.2 Main Menu ........................................................ 20
   3.3 Editor Module ...................................................... 22
       3.3.1 Ladder Diagram Data Structures ......................... 27
           3.3.1.1 Element List ..................................... 28
           3.3.1.2 Network List ..................................... 33
           3.3.1.3 Element Type Reference List .................... 34
           3.3.1.4 Small and Large Box Data Lists ............... 35

Table of Contents v
LIST OF ILLUSTRATIONS

Figure 1. Example of a Ladder Network and Equivalent Boolean Code 11
Figure 2. Functions Available in Each System Module ............ 21
Figure 3. Editor Menu Levels One and Two .................... 24
Figure 4. Editor Menu Level Three .......................... 25
Figure 5. Editor Menu Level Three (continued) ................ 26
Figure 6. Element Link Components ............................ 32
Figure 7. Example of a Ladder Network and its Nesting Levels .... 64
LIST OF TABLES

Table 1. DATA GROUPS STORED IN LADDER FILE ............ 42
CHAPTER 1. INTRODUCTION

There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things.

- Niccolo Machiavelli (The Prince, 1513)

In 1978, the National Electrical Manufacturers Association (NEMA) [1] defined a programmable controller (P/C) as "a digital electronic apparatus with a programmable memory for storing instructions to implement specific functions, such as logic, sequencing, timing, counting, and arithmetic, to control machines and processes." Bryan and Jones [1] describe a P/C as "a solid-state device used to control machine or process operation by means of a stored program and feedback from input/output devices." They further state that the single greatest benefit from the use of programmable controllers as control devices is that they are programmable. Once the P/C and its associated input/output (I/O) hardware are installed and the control program is stored in the memory of the P/C, the program can be modified to meet dynamic control requirements. This flexibility eliminates the need to design and install completely new hard wired control systems or perform major field wiring modifications whenever control changes are necessary.

A P/C based control system consists of two hardware components: 1) a central processing unit (CPU) and 2) I/O interface modules, and one software component: the control program. The CPU operates the control
program which implements the desired sequencing and command decisions. The I/O interface modules provide the connection between the software of the control program and the physical hardware to be controlled. When a control system is installed, the CPU and I/O modules are placed in a central location and communicate with the controlled equipment through field wiring. This hardware usually does not change unless new equipment to be controlled is added to the system or old equipment is removed. The control program, however, is typically modified more often than the physical system it controls due to product or process changes.

The P/C control program, then, can be considered the most critical of the three components, from the standpoint of flexibility. The successful operation of a system depends greatly upon the effectiveness of the system developers and users in designing and modifying the control program. That effectiveness, in turn, depends upon the ease of use as well as the sophistication of the programming tools used. These features are key to personnel productivity during the development or modification process.

Today, there exist many types of peripheral devices for use in programming programmable controllers. Bryan and Jones [1] provide the following list of the most common devices:

- Dumb Cathode Ray Tubes (CRTs),
- Intelligent CRTs,
- Mini-Programmers,
- Program Loaders,
• Memory Burners, and
• Computers.

Of the above devices, the intelligent CRTs and computers are the most powerful. They allow off-line programming in which the control program can be edited and stored without requiring that the programming device be connected to the P/C during the programming process. In addition, some CRT and computer programming systems provide documentation and reporting features, such as program comments, user-defined element names, and cross references. These features improve the programming process and make the programming devices efficient tools for editing, debugging, and interpretation of control programs. As an alternative to manual documentation procedures, they provide great savings in man-hours of preparation and modification.

1.1 PROBLEM STATEMENT

Currently there are more than fifty different manufacturers of programmable controllers. Many of them provide a broad line of controllers which span a wide range of sizes and control capabilities [2]. Additionally, each manufacturer markets equipment to program its controllers. One problem which exists in the P/C industry is the lack of compatibility between the controllers of different manufacturers and, in some cases, between the controller models of particular manufacturers. It is usually not possible to use the programming device of one manufacturer to program the P/C of another manufacturer. This problem is most often due to in-
compatible operating systems, but can also be attributed to the differences in the programming languages that are used.

Lack of standardization is not a problem in situations where a company owns only one P/C or several identical controllers. It is a problem, however, for those companies whose control requirements vary to the extent that different programmable controllers are needed for different machines or processes. In that case, the company must acquire and support two or more different programming devices. This requires additional time and effort on the part of the control engineers and technicians to become proficient with more than one device. In addition, alternating between different systems will always require some relearning of the operating details unique to each system and can possibly lead to programming errors.

What is needed by users of programmable control equipment is a universal programming device which can program a very wide range of controller products and provide extensive documentation utilities. Not only can this reduce the capital equipment requirements of an organization, but it can alleviate the personnel training and efficiency problems described above.

1.2 OBJECTIVE

The intent of this research encompasses the initial development of a universal programming system which will operate on a host IBM Personal Computer, Personal Computer XT, or Portable Personal Computer with 256K memory. (Henceforth, the universal programming system will be referred
to as the "system" or "system program".) To accomplish this, two phases of the research were defined: 1) design of an initial or core system and 2) investigation of the means to expand the initial system into a universal programming system.

The first phase included the design of an initial system program and its implementation on an IBM Personal Computer. The system is capable of developing ladder diagrams for a Texas Instruments (TI) Model 530 programmable controller using a subset of its ladder diagram instruction set and converting them to a Boolean code ready for download to the controller. The system design, however, allows for future expansion to the full TI Model 530 instruction set.

In developing the system program, emphasis was placed on the design of the human/computer interface. It was considered important that the system be practical and easy to use, so that a user can quickly learn how to operate it and be able to retain a good working knowledge of its operation after periods of non-use. It was of equal importance that the system be powerful enough to perform a wide variety of tasks so that the user can efficiently develop, modify, debug, communicate, and store control programs. The means to add future utilities such as entry/edit of ladder comments, global search for ladder elements, and generation of element cross references was designed into the program.

The objectives of the first phase can be summarized as the development of a programming/documentation system with the ability to:
• graphically create and modify ladder diagram programs,
• place comments in the ladder diagram program,
• identify ladder elements with alphanumeric labels,
• search for and display a particular rung or element in a ladder diagram program,
• save ladder diagram programs on a 5.25" floppy disk storage unit and retrieve those programs,
• download ladder diagram programs to a programmable controller,
• upload ladder diagram programs from a programmable controller,
• print a hard copy of a ladder diagram program on a printer attached to the personal computer, and
• create and print a cross reference for a ladder diagram program.

An investigation into the concept of a universal programming system and a feasibility study of such a system were conducted in Phase Two. This included a review of the software differences between programmable controllers that prevents the use of current programming devices in programming more than one controller or controller family. Once those differences were established, the means to overcome them was explored. Finally, the feasibility of expanding the initial system developed in Phase One to program other controllers was investigated.

1.2.1 Phase One Tasks

For the purposes of development, the system program was divided into the four major modules listed below:
1. Main Menu,
2. Ladder Diagram Editor (Editor module),
3. Store Ladder Program on Disk (Save module), and
4. Retrieve Ladder Program from Disk (Load module).

Operationally, the nucleus of the system program is the Main Menu, under which the three other modules are subfunctions. As the initial system expands to incorporate new features, the tasks will be added as additional menu options. For example, when the ability to program other programmable controllers is added to the system, a setup utility will most likely be required to configure the system for the appropriate instruction set and communications protocol. This utility would become another menu option.

Of the four modules listed above, the Editor module is the key component of the system for three reasons. First, it requires the greatest amount of human/computer interaction. Second, it has several subfunctions of its own. Some of these functions are listed in the summary of objectives above. Finally, as the system expands by adding to the instruction set and incorporating the program characteristics of other programmable controllers, the Editor module will undergo the greatest changes. Because of this importance, the editor design required the most attention during Phase One.

The following list summarizes the development tasks undertaken to complete Phase One:
• define the system program operation and general structure including available utilities, screen formats, and keyboard key definitions,

• develop an algorithm to convert displayed ladder diagrams into Boolean mnemonic language,

• design a data structure for ladder diagram program storage,

• develop an algorithm to convert programs stored in Boolean mnemonic language into the ladder display data structure,

• define all possible error conditions that may arise during system program operation and define error handling procedures,

• develop a library of 8088 assembler language utilities that may be called from Microsoft FORTRAN Version 3.2 and that provide the system program with color capabilities and access to the computer keyboard and screen,

• code the system program,

• test and debug the system program, and

• write a system program user's guide.

1.2.2 Phase Two Tasks

As discussed above, Phase Two established the needs of a universal programming/documentation system, determined the feasibility of its development, and proposed a development strategy. Much of the research in Phase Two depended upon the knowledge gained during Phase One. The general tasks required to accomplish the objectives of this phase are listed below:

• assess the feasibility and outline a strategy to expand the initial system to handle the entire TI Model 530 instruction set,
• obtain documentation on programming/documentation systems specific to programmable controllers manufactured by major vendors,

• identify the programming characteristics that are common to different programmable controllers and those that are different,

• investigate the problems associated with communication between the system and programmable controllers manufactured by different vendors,

• assess the feasibility and outline a strategy to expand the initial system to interface with other programmable controllers.

1.3 TERMINOLOGY

During the development of the system program, it became necessary to adopt a standard set of terms with which to refer to all of the components in a ladder diagram. In addition, new terms had to be created in order to describe the operation of the algorithms developed for this thesis. This section defines the terminology used throughout the paper. It is assumed that the reader understands how basic ladder diagram elements such as contacts, coils, timers, and counters function and that he or she is familiar with the fundamental operation of a programmable controller.

A ladder diagram is a representation of logical decisions and instructions shown between two vertical rails representing opposing polarity. Each individual instruction in a ladder diagram is called a ladder element or, simply, element. The diagram is divided into separate groups of elements called networks. Each network performs one or more discrete operations such as receiving input signals from the controlled equipment, sending
output signals, or executing operations internal to the controller. A network is constructed by forming horizontal rows of elements in series, called rungs, which connect the two rails. Two or more rungs may be connected in parallel in the same network. Figure 1 on page 11 shows an example of one network composed of nine elements on two rungs.

If a rung is connected to another parallel rung, then it will be divided into rung segments by the vertical connections linking it to the other rung. Vertical connections are also referred to as OR links. In Figure 1, for example, the portion of rung one on which contact X1 is located or the portion of rung two on which contacts X5 and X6 are located are rung segments. In other words, a rung segment is any portion of a rung that is bounded on both sides by vertical connections to a rung above or vertical connections from a rung below.

When a network is displayed by the Editor module of the system program, as many as twelve elements may be placed in series and up to seven rungs may be connected in parallel. For editing purposes, the screen is divided into a matrix of seven rows and twelve columns. The first eleven columns are reserved for input elements such as contacts, and the twelfth column is reserved for output elements such as coils or control relays. Each subdivision in the matrix, called a cell, can contain one element. Some of the more advanced instructions are displayed by the editor as a small box or large box because they must show more information than basic elements like contacts or coils. Small boxes cover nine matrix cells in
Figure 1. Example of a Ladder Network and Equivalent Boolean Code

BOOelan CODE

STR X1
OR X2
STR X3
AND X4
STR X5
AND X6
OR STR
AND STR
AND NOT X7
OUT Y8
OUT NOT Y9

CHAPTER 1. INTRODUCTION
three rows and three columns. Large boxes cover sixty-three matrix cells in seven rows and nine columns.

To describe the location of a basic element on the editor display, its coordinates may be simply stated by row and column. This is more difficult for a box, however, since it covers more than one matrix cell. The reference to the coordinates of a box will mean the coordinates of the upper left corner of that box (the home cell). The coordinates of a box, then, will be referred to as the home row and home column of that box.

When a network is displayed by the editor, that network will be called the current network of the ladder. Similarly, wherever the editor cursor is located will be called the current row and current column. If the cursor also happens to be positioned on an element in a network, that element will be referred to as the current element and the rung on which it is located will be the current rung.
2.1 THE HISTORY OF PROGRAMMABLE CONTROLLERS

Bryan and Jones [1] provide the most comprehensive discussion on the historical background of programmable controllers. They attribute the Hydramatic Division of General Motors Corporation with the initial development of programmable controllers, first specifying the design criteria in 1968. The impetus behind GM's efforts was the high cost associated with inflexible, relay-operated control systems for the large machining lines in its automobile production plants. Constantly plagued by machine down time and the inability to easily expand or modify the control systems, GM decided to create a solid-state system with computer flexibility which could not only survive harsh industrial environments, but could be easily programmed and flexible enough to allow timely modification and future expansion.

The first programmable controllers did nothing more than replace relay-operated systems and could only process digital input and output. Therefore, they were limited to the on/off control of repetitive type operations such as transfer lines. Nevertheless, they were clearly an improvement over the conventional control systems. In addition to the advantages of flexibility, reliability, and ease of use, the P/C control systems required much less space and energy.
As time passed, several innovations improved the capability of programmable controllers. As a result, the number and types of applications in which they could be used increased rapidly.

2.1.1 Hardware Developments

Advances in microprocessor technology allowed the controllers to become faster and more intelligent. Greater processor speed meant that faster operations could be controlled. In addition, better processors permitted the expansion of programmable controller memory, thereby allowing the storage of larger control programs and more data. Greater intelligence and increased memory also made it possible for programmable controllers to perform more elaborate tasks such as data acquisition and manipulation.

Improved processors also made it possible to add analog control to programmable controllers, bridging the gap between on/off control systems and instrumentation controls. With this capability, programmable controllers could totally manage processes which required both discrete and variable control functions such as chemical batching, and water and waste treatment.

Other hardware developments included the ability to perform positioning control for stepper motors. The P/C input interface counted a train of incoming pulses from the motor, converted it to a value, and passed it to the P/C. The P/C evaluated the data and passed the required motor control data to the output interface. The output interface then produced
a pulse train to be interpreted by the stepper motor translator. Early application of this positioning control capability included grinders, transfer heads, and paint spray lines.

However, much of the hardware development in programmable controllers was in communications. Early improvements introduced off-line programming devices and the ability to generate production summaries, management reports, and maintenance data on an associated printer.

The original programmable controllers and their associated I/O hardware which interfaced the controlled machines or processes had to be placed together at one central location. Signals to and from the P/C were carried over long wiring runs. The advent of intelligent input/output subsystems made it possible to place the interface hardware in remote locations away from the P/C, significantly reducing wiring costs. Instead of running hundreds of wires back to the P/C, the signals from each subsystem could be multiplexed over a twisted pair of wires.

Current developments in programmable controller communications include the ability to tie several controllers together into one high-speed distributed network or data highway. These networks can also be tied into a large host computer for central control of an entire plant. The advantage of this plant-wide integration is that real-time production data can be captured and processed by the production control, material/resource planning, and management information systems.
2.1.2 Software Developments

The first improvements in programmable controller software expanded the instruction sets. The addition of arithmetic and data manipulation functions allowed programmable controllers to be used with instrumentation devices that provided numerical input data. For example, now a P/C can perform calculations based on measured data and report meaningful output information instead of raw data. These software enhancements also allowed programmable controllers to operate under changing system conditions. Thus, if a certain event occurred, a P/C could automatically access previously stored data and change timer and counter preset values accordingly.

The introduction of analog interface hardware brought about the need for higher level languages that provided computer-like statement instructions. Communicating with this type of hardware would have been difficult or impossible with only relay-type instructions.

Other software improvements provided system routines to improve on-line monitoring of a process. Also, today's programmable controllers are capable of performing system diagnostics to check for controller malfunctions, and machine diagnostics, for failures or malfunctions of the controlled equipment.

CHAPTER 2. LITERATURE REVIEW
2.2 THE FUTURE OF PROGRAMMABLE CONTROLLERS

Bryan and Jones [1] state that the most important changes in the programmable controller industry will be continuations of the following current trends:

- improved microprocessors,
- intelligent I/O interfaces,
- distributed and hierarchical control within integrated systems,
- high level language developments, and
- improved diagnostic capabilities.

The programmable controllers of the future will be integrated with other control equipment such as robots, numerical controls, CAD/CAM systems, and management information systems to achieve higher levels of product quality and increased productivity.

2.3 UNIVERSAL PROGRAMMING/DOCUMENTATION SYSTEMS

Literature discussing previous efforts to develop universal programming/documentation systems for programmable controllers is currently unavailable. In fact, very little literature could be found in professional publications regarding P/C programming/documentation devices or systems at all. Since the majority of previous programmable controller research has been conducted by commercial organizations, any literature which does exist is proprietary. There is, however, some information...
available in marketing literature from several companies which provides an indication of the extent to which current systems are universal. These brochures also give information about the various programming utilities that are available with some of the systems.

A few programmable controller manufacturers advertise programming devices which can program P/C products from one or two other manufacturers as well as their own products. The only programming/documentation product not sold by a programmable controller manufacturer is available from Process and Instrumentation Design, Incorporated (P&ID) [18]. P&ID markets a system for the entire line of Allen-Bradley controllers. This product runs on a Fortune 32:16 microcomputer with a Unix operating system. It allows off-line programming and features an extensive set of menu-driven utilities. Its various output options include a ladder diagram program listing, data tables, cross reference tables, and a summary of unused elements. The program editor allows program and section titles, a program comment page, and comments above, below, and at one side of each rung. In addition, each element may be given a five line by ten character label. The editor also features search and replace utilities. Ladder programs may be stored on and retrieved from diskettes or tape, but cannot be directly uploaded to or downloaded from a programmable controller. In summary, the system is very extensive and well designed, however, it can only interface with one brand of programmable controller.

Other third party systems are capable of interfacing with more than one brand of programmable controller, but are documentation tools only and
cannot be used to program the controllers. Four companies from which literature was obtained are Taylor Industrial Software [19], Emtrol [20], Automation Consulting Services [21], and Indelec [22]. The primary function of these systems is to upload a ladder diagram program from a P/C, display the program for editing and adding comments, and then print a copy of the program or save it on a diskette. They are not capable of downloading new or modified programs to a P/C. The advantage in using these systems is that they provide more extensive documenting facilities than the P/C manufacturers' programming devices.
3.0 CHAPTER 3. METHODOLOGY

3.1 SYSTEM OVERVIEW

The system program consists of four major modules. Those modules are the Main Menu, the Editor, the Save Module, and the Load Module. The main menu provides access to all of the system's major modules. The editor displays ladder diagrams and allows them to be created and modified. It uses a series of menus for selecting the editing functions. The save and load modules each perform one task. They supply the ability to store and retrieve programmable controller programs. The programs may be stored in either a Boolean code format or a ladder diagram data format. Since an incomplete ladder diagram cannot be converted to Boolean code, the second file format was created to store the incomplete programs for later editing. Figure 2 on page 21 is a listing of all tasks that are available in the Editor, Save and Load Modules. The sections which follow discuss the operation of all four major modules. The chapter concludes with a discussion of the man-machine interface.

3.2 MAIN MENU

Operationally, the main menu is the nucleus of the system program. From it, all of the other major modules may be called. The menu displays the names of all the system modules in a vertical list. A flashing arrow, located just to the left of one of the module names, indicates the current
EDITOR MODULE:
Menu Options:
   return to main menu
   edit an element
   add a small box element
   add a large box element
   add a vertical connector
   add a horizontal element
   add a normally open contact
   add a normally closed contact
   add a normally open coil
   add a normally closed coil

Movement Functions:
   move cursor up
   move cursor down
   move cursor left
   move cursor right
   move cursor to column one
   move cursor to column twelve
   move cursor to row one
   move cursor to row seven
   display first network
   display last network
   display preceding network
   display following network

Other Functions:
   insert a network
   delete a network
   delete an element

SAVE MODULE:
   return to main menu
   save ladder file
   save Boolean file

LOAD MODULE:
   return to main menu
   load ladder file
   load Boolean file

Figure 2. Functions Available in Each System Module
menu option. Above the list is a message instructing the user to press either the cursor up key, the cursor down key, or the carriage return (enter) key. Below the list is a one line description of the current option.

Pressing the cursor up key causes the arrow to move up one option. Pressing the cursor down key causes the arrow to move down one option. Finally, pressing the carriage return key causes the system to execute the option that was indicated by the arrow. Another method to select one of the options is to press the alphabetic character key which corresponds to the first letter of the option name. Thus, to use the editor, the user would either position the arrow next to editor option and press the carriage return key or simply press the E key.

If the user presses any key that is not expected by the system, it will display the message 'Invalid Key' at the bottom of the screen and allow the user to try again.

3.3 EDITOR MODULE

The editor provides the user with a means to graphically construct and modify ladder diagrams for later downloading to a programmable controller.

The editor module screen is divided into three areas. The first area is the network display area. In it, each network in a ladder diagram may
be displayed for editing. This area covers lines one through twenty-two of the twenty-five line screen. The next area is the menu area, which lists the editor functions on lines twenty-three and twenty-four. The last area on the screen is the status line on line twenty-five. It displays information for the user concerning the ladder diagram being edited. On the left side of the status line is the current network number and the coordinates of the cursor. On the right are indicator flags to inform the user that the last ladder in the network is currently displayed or that the ladder cannot accept any more elements.

The editor uses a menu system which consists of three menu levels. This design allows the editor to provide a large number of tasks that can be controlled using only the ten function keys at the left side of the keyboard. Figure 3 on page 24, Figure 4 on page 25 and Figure 5 on page 26 show the function key assignments that would be used by the system for programming the TI Model 530 programmable controller with its full instruction set. Only a small subset of these key assignments are used by this initial system.

Upon initiation, the editor module takes one of two actions depending on whether there is a ladder file in memory or not. If there is a file in memory, then the editor calls subroutine SHOWNW, which displays the first network in the ladder. If there is no file in memory, then the editor calls subroutine INITNW. This subroutine displays an initialized network which contains one rung consisting of one normally open contact, one normally deenergized coil, and eleven horizontal elements connecting

CHAPTER 3. METHODOLOGY
### LEVEL ONE:

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MENU</td>
<td>EDIT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALL BOX</td>
<td>LARGE BOX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-/ /-</td>
<td>-(/)-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-( )-</td>
</tr>
</tbody>
</table>

### LEVEL TWO:

**PREVIOUS KEY:** F3 SMALL BOX

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTER</td>
<td>TIMER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATH</td>
<td>LOGICAL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>WORD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>O/S</td>
<td>CONVERT</td>
</tr>
</tbody>
</table>

**PREVIOUS KEY:** F4 LARGE BOX

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT DRUM</td>
<td>DRUM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCAN MATRIX COMPARE</td>
<td>INDEXED MATRIX COMPARE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y coil</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C coil</td>
</tr>
</tbody>
</table>

**PREVIOUS KEY:** F8 -(/)- or F10 -( )-

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y coil</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PREVIOUS KEY:** F7 -/ | - or F9 - | -

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5</th>
<th>F6</th>
</tr>
</thead>
<tbody>
<tr>
<td>X contact</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y contact</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F9</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C contact</td>
<td></td>
</tr>
</tbody>
</table>

---

Figure 3. Editor Menu Levels One and Two
### LEVEL THREE:

<table>
<thead>
<tr>
<th>Previous Key: F3 Counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
</tr>
<tr>
<td>F3 COUNTER</td>
</tr>
<tr>
<td>F5</td>
</tr>
<tr>
<td>F7</td>
</tr>
<tr>
<td>F9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Previous Key: F5 Math</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
</tr>
<tr>
<td>F3 ADD</td>
</tr>
<tr>
<td>F5 MULTIPLY</td>
</tr>
<tr>
<td>F7 SQUARE ROOT</td>
</tr>
<tr>
<td>F9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Previous Key: F6 Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
</tr>
<tr>
<td>F3</td>
</tr>
<tr>
<td>F5</td>
</tr>
<tr>
<td>F7</td>
</tr>
<tr>
<td>F9</td>
</tr>
</tbody>
</table>

Figure 4. Editor Menu Level Three

CHAPTER 3. METHODOLOGY
LEVEL THREE (continued):

<table>
<thead>
<tr>
<th>PREVIOUS KEY:</th>
<th>F7 BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F2</td>
</tr>
<tr>
<td>F3</td>
<td>F4</td>
</tr>
<tr>
<td>F5</td>
<td>F6</td>
</tr>
<tr>
<td>F7 BIT PICK</td>
<td>F8</td>
</tr>
<tr>
<td>F9 BIT SET</td>
<td>F10</td>
</tr>
<tr>
<td></td>
<td>BIT SHIFT REGISTER</td>
</tr>
<tr>
<td></td>
<td>BIT CLEAR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PREVIOUS KEY:</th>
<th>F8 WORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F2</td>
</tr>
<tr>
<td>F3 WORD SHIFT REGISTER</td>
<td>F4</td>
</tr>
<tr>
<td>F5 MOVE WORD TO IR</td>
<td>F6</td>
</tr>
<tr>
<td>F7 MOVE WORD</td>
<td>F8</td>
</tr>
<tr>
<td>F9 MOVE WORD TO TABLE</td>
<td>F10</td>
</tr>
<tr>
<td></td>
<td>MOVE DATA CONSTANT</td>
</tr>
<tr>
<td></td>
<td>MOVE WORD FROM TABLE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PREVIOUS KEY:</th>
<th>F9 CONVERT</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>F2</td>
</tr>
<tr>
<td>F3</td>
<td>F4</td>
</tr>
<tr>
<td>F5</td>
<td>F6</td>
</tr>
<tr>
<td>F7</td>
<td>F8</td>
</tr>
<tr>
<td>F9 CONVERT BINARY TO BCD</td>
<td>F10</td>
</tr>
<tr>
<td></td>
<td>CONVERT BCD TO BINARY</td>
</tr>
</tbody>
</table>

Figure 5. Editor Menu Level Three (continued)
them. This configuration is the minimum requirement for a ladder network. The contact and coil do not have reference numbers as these must be assigned by the user. After returning from either of these two routines, the editor enters its menu mode and waits for the user to select a function. This section describes the two most important aspects of the editor module: the ladder diagram data structure design and the algorithm used to display a ladder network.

3.3.1 Ladder Diagram Data Structures

The ladder diagram data structures are the heart of the editor module. All of the editor functions are affected by their design. The primary objective of the data structures is to store information on each element in a ladder diagram and allow the system to display those elements. It is important to note at this point that a Boolean instruction, when discussed in the context of a ladder diagram, will be referred to as a ladder element or element. For example, the Boolean instruction AND X1 is represented in a ladder diagram as a normally open series contact. That contact is a ladder element.

The data structures were designed to meet the immediate needs of the initial programming system. Those needs include ladder display, ladder editing, ladder to Boolean conversion, Boolean to ladder conversion, and ladder storage and retrieval. The structures were also designed in anticipation of future expansion of the system to include search, replace
and cross reference functions, ladder comments, element labels, ladder
printouts, and upload and download operations.

The primary data structure for this system is the element list. In ad-
dition to that structure, there are the following supporting data struc-
tures:

1. Network List (array NETWRK)
2. Element Type Reference List (array TYPREF)
3. Small Box Data Lists (array SBOX and SBOXC)
4. Small Box Cross Reference List (array SBXREF)
5. Large Box Data Lists (array LBOX and LBOXC)
6. Large Box Cross Reference List (array LBXREF)

Appendix A contains tables which show the format of each data structure.

3.3.1.1 Element List

The memory capacity of a TI Model 530 programmable controller allows
storage for up to 4,096 individual elements in 2,048 networks. To reach
this capacity all elements would have to be contacts, coils, or control
relays with one input and output element on each network. The element
list was designed to accommodate that limit.

A linked rather than sequential linear list data structure was chosen for
the element list for a number of reasons. First, the editor must be able
to add and remove elements from the element list. Page and Wilson [13] state that inserting and deleting records with a linked list is easier, hence faster, to do since the operations only require small changes in the links. With sequential storage, these operations may require considerable reorganization of storage and shifting of elements. Second, the most common system operation which requires access to the list is network display. Both linked storage and sequential storage are equally efficient in performing sequential access, however, sequential storage provides much faster random access. As will be shown in the next section, the display algorithm accesses the list sequentially whenever a new network is displayed. Other operations, such as search and replace, require random access, but their use was judged to be infrequent relative to the display operation. Therefore, linked storage was judged to be adequate with regard to list access. A third consideration was data storage requirements. Although the linked list requires more storage space than a sequential list, algorithms using sequential storage can become very inefficient when they approach high density loading. This could be a problem for very large ladder diagrams. To summarize, the ability to easily insert and delete records with linked lists was considered to override the disadvantages associated with greater storage space and list access. Therefore, a linked storage scheme was selected. In addition, the list is doubly linked, with pointers to both preceding and following records. This provides more flexibility than a singly linked list.

The element list consists of two arrays: ELMNTC, a character array, stores data in one byte fields and ELMNTI, an integer array, stores data.

CHAPTER 3. METHODOLOGY
in two byte fields. As much of the element information as possible was placed in one byte fields to conserve storage space. Each record of the character array consists of four fields. These fields hold information on the element type, its row coordinate, its column coordinate, and its physical links to other ladder elements. Appendix A contains a table of code values developed for storing the element types of TI Model 530 instructions. Each record of the integer array consists of three fields. The first two fields are backward and forward pointers respectively. For all records in the list, except the first and last, the backward pointer points to the preceding record and the forward pointer points to the following record. The backward pointer of the first record is set to zero and the forward pointer of the last record is set to minus one. The last field holds a value which depends upon the element type. If the element is a contact or coil, then field three stores a user specified reference number for the element. If the element is a small or large box, then the field holds a pointer to the small or large box data lists. Both the character and integer arrays use the same index numbers. For example, the fifth record of the character array and the fifth record of the integer array hold information on the same element.

There are six supporting variables for the element list. ELHEAD points to the first record in the list, while ELFREE points to the the first unused storage locations in the arrays. ELETOT stores the current number of records in the list and ELEMAX holds the maximum number of storage positions in the list (the size of the character and integer arrays). ELEPTR, which is discussed in the display algorithm section, holds
pointers to the element list. Finally, ELFULL is a logical variable which indicates whether or not the element list is full of records.

One final aspect of the element list which requires discussion is the element link data stored in field four of the character array. Every input element in a network has two physical links which connect it to the elements on either side of it and the rungs above and below it. These will be called the left link and right link. For the purposes of this discussion, consider the links on the left and right to be nodes which are located between the element and its adjoining elements. From the node, there can extend up to four spokes which will be called link components. These link components are assigned unique values and connect the node to the element on its left, the element on its right, the rung above, and the rung below.

Figure 6 on page 32 shows the link component configuration and the values assigned to each component.

Since each node has four components, there are sixteen possible arrangements for the components. Therefore, there are sixteen possible link configurations on either side of an element. These link configurations will be called link types and will be numbered from one to sixteen. Since a given input element will have a unique link configuration on its left and a unique link configuration on its right, there will always be two link types associated it. These are the left link type and the right link type. Appendix A contains a table which lists all of the left and right link types. The table also shows examples of elements (marked by a square
Figure 6. Element Link Components
bullet) which possess each of the link types. Output elements differ slightly from input elements in that they only have a left link. There are four possible link types for output elements. These are also listed in Appendix A.

3.3.1.2 Network List

All records in the element list are grouped by network. Thus, all elements in a given network are linked sequentially from the first element in the network to the last element. (The concept of element order within a network is discussed in the display algorithm section.) The first element in the network is linked to the last element in the preceding network. Similarly, the last element in the network is linked to the first element in the following network. The network list provides access to the element list by network and allows up to 2,048 networks to be stored.

The structure of the network list is similar to the element list for the same reasons described above. The network list, however, consists of only an integer array with five two-byte fields. Fields one and two are pointers to the element list. Field one points to the first element in a network and field two points to the last element in a network. The third field stores the number of rungs in a network. Fields four and five are backward and forward pointers. Field four points to the preceding network record in the list and field five points to the following network record. As in the element list, the backward pointer of the first record is set to zero and the forward pointer of the last record is set to minus one.
There are seven supporting variables for the network list. NWHEAD points to the first record in the list and NWFREE points to the first unused storage location in the array. NTWTOT stores the current number of records in the list and NTWMAX holds the maximum number of storage positions in the array. NTWPTR, points to the current network record and NWTAIL points to the last record in the list. Finally, NWFULL is a logical variable which indicates whether or not the network list is full of records.

3.3.1.3 Element Type Reference List

The element type reference list stores the type of a contact or coil element. It is used by the editor to prevent the user from assigning the same reference number to both an input (X type) and an output (Y type) element simultaneously. The list holds 1,023 records, which is the maximum number of input and output elements allowed by the TI Model 530 controller. Access to the list is by user specified reference number. As an example, if the user places a normally open input contact in a network and assigns it a reference number of 150, then an 'X' is stored in record 150 of the element type reference list. Because random access to the list is required and no insertion operations are performed on the list, a sequential linear list was chosen as the data structure.
3.3.1.4 Small and Large Box Data Lists

The small and large box data lists hold additional information on their respective boxes that cannot be stored in the element list. Each consists of a integer (two-byte) array and a character array. The small box data list can hold information on 128 small boxes and the large box data list holds information for 30 large boxes. These values are the maximum allowed by the TI Model 530 controller. As it is for the element type reference list, access to these two lists is random and no insertion operations are performed, therefore these are both sequential linear lists.

3.3.1.5 Small and Large Box Cross Reference Lists

The small and large box cross reference lists were created to allow a search operation in the editor for boxes. Access to the lists is by box reference number. Both lists contain only one field and the value stored is a pointer to the box element data in the element list.

The cross reference is not necessary for elements other than boxes because their reference numbers are stored in the element list. If the user wished to locate all contacts with a given reference number, the search algorithm would examine field three of each record in the element list integer array until it found the desired contact or contacts. This simple method is not possible for boxes since their reference number is stored in the box data lists. To find a box with a particular reference number,
the search algorithm would refer to the cross reference list to obtain the pointer to the element list for the desired box.

3.3.2 Display Algorithm

The editor uses the display algorithm whenever it displays a particular network. The subroutines associated with the algorithm are SHOWNW, WRELEM, WRLNKL, WRLNKR, and WRTDATA. Subroutine INITNW, mentioned earlier, is similar to SHOWNW and performs a modified version of this algorithm.

The algorithm works by reading the information on each element in a network and displaying it on a network "map" or matrix on the CRT screen. The matrix spans seven rows and twelve columns and contains ninety-four cells. Each row corresponds to a network rung and each column corresponds to one element position on the rungs. Input elements may be placed on any row from column one to column eleven while output elements may only be located in column twelve on any row. Contacts and coils each take up one row and one column to fill one matrix cell. Boxes, because of the additional information they must display, require more space on the screen than one matrix cell. Small boxes use nine matrix cells in three consecutive rows and columns. Large boxes take up almost the entire screen with sixty-three cells in seven rows and nine columns.

The order of network elements is determined by their position on the matrix. It is a row major arrangement. If each matrix cell contained one
element, then the first element in the network would be the one at row one, column one. The second element would be the one at row one, column two. This order continues through column eleven where the next element would be the one at row two, column one. Thus, all of the elements in columns one through eleven (input elements) precede the elements in column twelve (output elements). The order of output elements is from row one to row seven, so the last element in the network would be the one at row seven, column twelve.

The two primary variables used by the algorithm are ROWCNT and ELEPTR. ROWCNT is an array which holds the number of elements on a rung for each of the seven rows. ELEPTR, a two dimensional array whose elements correspond to the matrix cells, stores pointers to the element list for every element located on the matrix. For small and large boxes, the same pointer is stored in each matrix cell covered by the box. Empty cells (those which do not contain an element) are assigned a negative value. The value is always that of the element list pointer for the element which precedes the empty cell. This value will be called a "back pointer" and facilitates proper ordering of new elements as they are added to the network. For example, consider the following arrangement. The cell at row one, column one contains an element whose information is held in record number ten in the element list. The cell at row one, column two does not contain an element. In this case, the value stored in element (1,1) of array ELEPTR would be ten, whereas the value stored in array element (1,2) would be minus ten. If a new element were to be added to the network
The display algorithm performs the following steps each time a network is displayed:

1. The first step of the algorithm clears the network display area and displays the network number on the status line. If the network to be displayed is the last network in the ladder, then the message 'ladder end' is placed on the status line. Then, arrays ROWCNT and ELEPTR are initialized to zero.

2. Next the element list pointer for the first element in the network is obtained from the network list.

3. The row and column position for the element is obtained from the element list and the element list pointer is then assigned to the correct matrix cell. If the element is a box, then the pointer is also assigned to the other matrix cells covered by the box.

4. The algorithm then calls subroutines WRELEM and WRDATA. WRELEM draws the element on the screen and then calls subroutines WRLNK and WRLNK to draw the links to the elements on either side of it. WRDATA displays any information associated with the element, such as the element reference number.

5. If the element just displayed is not the last element in the network then the algorithm obtains the element list pointer for the next element and returns to step three. Otherwise, the algorithm proceeds to step six.

CHAPTER 3. METHODOLOGY
6. Once all elements have been displayed, the back pointers are assigned
to all empty matrix cells.

7. Finally, the algorithm sets the current row and column, for editing
purposes, to one and highlights the screen position at row one and
column one.

Once step seven is done, the algorithm ends. Subroutine SHOWNW then re-
turns to the EDITOR subroutine to await a command from the user.

3.4 SAVE AND LOAD MODULES

The save and load modules allow the user to store and retrieve both ladder
diagrams and the Boolean code for the diagrams on diskettes. These mod-
ules are performed by subroutines SAVE and LOAD.

When either the SAVE or LOAD options are selected on the system main menu,
the system displays the Save Screen or Load Screen and awaits one of two
inputs from the user. The inputs are file type and file name.

Near the top of the screen the menu displays two labels: Ladder File and
Boolean File. One of the labels is highlighted in reverse video. Ladder
diagrams are stored in a ladder file and Boolean code is stored in a
Boolean file. Each file type has its own unique file name extension.
All ladder files use the extension .LDR and all Boolean files use the
extension .BOL. To select the file type, the user presses the tab key.
Each time this key is pressed, the highlight moves from one label to the other.

Below the labels is an input field which allows up to eight characters to be entered. A prompt next to the field instructs the user to enter the file name of the desired file. To enter a file name, the user simply types in the name of the file. The file name may be from one to eight characters long. The file extension should not be entered in this field, since it is automatically added by the program according to the selected file type. If the user attempts to enter a character that is not allowed in IBM PC file names, then the system will display the message 'Invalid Key' and allow the user to try again. If eight characters are entered in the field, then a flashing carriage return symbol appears next to the input field to remind the user that the field is full and that his next action should be to press the carriage return key. When the carriage return key is pressed the file is saved or loaded.

3.4.1 Save Module

When the SAVE option is selected and the user specifies that a Boolean file is to be saved, the system uses the ladder to Boolean conversion algorithm to convert the ladder diagram data in memory to Boolean code and save it on the diskette. The conversion algorithm is discussed later in this chapter. When the user specifies that a ladder file is to be saved, then the system stores the ladder diagram data directly on the diskette. This section describes the saving process.
The first step in the process is to open a file on the diskette with the file name specified by the user. Then, two check values are written to the file. The save routine writes four file check values; two at the beginning of the file and two at the end. In addition to marking the beginning and end of the file, they allow the load module to check that a file is a valid ladder file and check that all data has been read correctly.

After the check values are written, the system begins writing ladder data to the file. The data is written by group in the order listed in Table 1 on page 42. Prior to writing each data record, the system writes a record check value that is unique to its group. This is another means to ensure that the file is free from errors when it is read by the load module. The check values for each group are also listed in the table. After the system writes all of the ladder data to the disk, it writes the final file check values and returns program control to the SAVE menu.
Table 1. DATA GROUPS STORED IN LADDER FILE

<table>
<thead>
<tr>
<th>GROUP</th>
<th>CHECK VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element Data</td>
<td>4</td>
</tr>
<tr>
<td>Network Data</td>
<td>5</td>
</tr>
<tr>
<td>Element Type Reference Data</td>
<td>6</td>
</tr>
<tr>
<td>Small Box Data</td>
<td>7</td>
</tr>
<tr>
<td>Large Box Data</td>
<td>8</td>
</tr>
</tbody>
</table>
3.4.2 Load Module

The load module operates similarly to the save module. Before it opens the file and reads data, however, it calls subroutine INITLL to initialize all of the ladder diagram data structures. When the LOAD option is selected on the system main menu and the user specifies that a Boolean file is to be loaded, the system uses the Boolean to ladder conversion algorithm to read the Boolean code and convert it to ladder data. This conversion algorithm is also discussed later in the chapter. When the user specifies that a ladder file is to be loaded, then the system reads the ladder diagram data directly from the diskette.

Ladder data is read from the file in the same order that it was stored. First the two check values are read and compared with the correct values. Next, all ladder data records are read from the file. Each time a record is read, the system compares the record check value with the correct group value. Finally, after the end of file check values are read and tested, the system closes the file and returns to the LOAD menu.

If at any time during the load process an error is found, the system reports the error and closes the file. It then calls subroutine INITLL to initialize the ladder data structures and returns to the main menu.
3.5 LADDER TO BOOLEAN CONVERSION ALGORITHM

The algorithm to convert ladder display data to Boolean code is performed by subroutines DISBOL, PASS1, PASS2, and PASS3. It operates on one network at a time and accomplishes its task in three passes. Pass one builds an ordered list, known as a rung list, from the display data for each rung in a network. Pass two uses the rung lists to determine the root rungs for each rung segment. It then adds this information to the appropriate rung list. Pass three produces the Boolean code from data stored in the rung lists and writes the code to a disk file. All three passes check for errors in the ladder diagram and report if any are detected. This section describes how the conversion algorithm works and discusses the error detection and reporting scheme used.

3.5.1 Data Setup

To convert each ladder network into Boolean code, the data describing each network must be extracted from the element list and mapped onto a network matrix as described in the section on network display. The variable ELEPTR(I,J) is again used to hold network matrix data. For this routine, however, the linked list pointers assigned to each matrix cell may be positive, negative, zero, or the value 32,767. A positive value indicates a normally open contact, normally deenergized coil, or box element. A negative value indicates a normally closed contact or a normally energized coil. A zero value indicates that no element is present and the value 32,767 indicates that a horizontal element is present at that position.
Type 2 elements and boxes are also handled differently than in the display routine. No linked list pointer is assigned to matrix cells containing type 2 elements, since these have no equivalent in Boolean code. For small boxes, rather than assign their linked list pointer to all of the nine matrix cells they cover, the pointer is assigned only to cells in the first column covered by the box. In addition, the type of small box governs which cells in the first column are assigned the value. For single input boxes, only the uppermost cell receives the pointer value. For two-input boxes, the uppermost two cells receive the pointer value. Finally, for three-input boxes, all three cells in the first column are assigned the pointer value. As an example, a two input box located at position (I,J) would encompass matrix cells (I,J) to (I,J+2), (I+1,J) to (I+1,J+2), and (I+2,J) to (I+2,J+2) in the display algorithm. For the conversion algorithm, the linked list pointer would be assigned only to cells (I,J) and (I+1,J). Large boxes are handled in a similar manner to the small boxes. The linked list pointer is assigned only to the uppermost cell in the first column covered by the box. All other cells that would be associated with the box in the display algorithm are not assigned the pointer value.

In addition to the above information, additional data is stored in the matrix to mark the presence of vertical connectors in the network. The logical variable NODFLG(I,J) is assigned a TRUE value if the element in matrix cell (I,J) possesses an OR link on the left (left link value 64). In situations where an element possesses an OR link on the right, but no element is present on the immediate right, the above assignment will
overlook the right link. To prevent such omissions, NODFLG(I,J+1) is assigned a TRUE value if the element in matrix cell (I,J) possess an OR link on the right (right link value 4). In most cases this is a redundant operation. At the very beginning of the algorithm, the array NODFLG is initialized so that all values except those for column one are FALSE. NODFLG array elements for column one are initialized to TRUE since all elements in this column are automatically ORed with rung one.

3.5.2 Pass One

The first pass of the conversion algorithm examines the data for each matrix cell and builds a rung list for each network rung. The rung list information is stored in array LIST. Each rung list may contain up to 155 records each consisting of four fields. Field one is assigned a value which indicates the record type. The remaining fields contain data which varies by record type. Table 2 summarizes the data held by each record type. There are five types of records as follows:

Record Type 1:: The first record type contains information about the elements in the network. Field one contains the value 1. Field two contains the linked list pointer for the element and can be positive, negative, or zero. If the element is a normally open contact, normally deenergized coil, or a box, then the value in field two is positive. If the element is a normally closed contact or normally energized coil, then the value is negative. If the element is a horizontal, then the value
is zero. Field three contains the column position of the element. Field four is not used.

Record Type 2:: The second record type contains rung node information. Field one contains the value -1. Field two contains the rung node number and will be a value from one to twelve. Field three describes the node type and can be one, zero, or minus one. If the rung node marks the beginning of a rung segment, then the value in field three is one. If the rung node is a pass through node, then the value is zero. If the rung node marks the end of a rung segment, then the value is minus one. Field four identifies the rung segment on which the node is located. It can be a value from one to twelve.

Record Type 3:: The third record type contains connecting node information and indicates the end of a connecting parallel segment on a lower rung. Field one contains the value -2. Field two contains the rung number of the connecting parallel segment. Fields three and four are not used.

Record Type 4:: The fourth record type also contains connecting node information. It indicates the beginning of a connecting parallel segment on a lower rung. Field one contains the value -3. Field two contains the rung number of the connecting parallel segment. Fields three and four are not used.
**Record Type 5:** The last record type indicates the input to a small box and contains information about the box. Field one contains the value -4. Field two identifies the box type. For a single input box, the value in field two is zero. For a two-input box, the value is one. For a three-input box, the value is two. Field three contains the rung position of the box. Field four contains the column position of the box.
Table 2. Summary of Record Types for the Ladder to Boolean Conversion Algorithm

RECORD TYPE 1: LADDER ELEMENT RECORD
FIELD 1: 1
FIELD 2:
> 0 pointer to element linked list for a normally open contact, normally deenergized coil, small box or large box
0 horizontal
< 0 pointer to element linked list for a normally closed contact or normally energized coil
FIELD 3: column position of element
FIELD 4: not used

RECORD TYPE 2: RUNG NODE RECORD
FIELD 1: -1
FIELD 2: node number
FIELD 3:
1 beginning of rung segment
0 pass through node
-1 end of rung segment
FIELD 4: segment number

RECORD TYPE 3: END CONNECTING NODE RECORD
FIELD 1: -2
FIELD 2: rung number of connecting parallel segment
FIELD 3: not used
FIELD 4: not used

RECORD TYPE 4: BEGIN CONNECTING NODE RECORD
FIELD 1: -3
FIELD 2: rung number of connecting parallel segment
FIELD 3: not used
FIELD 4: not used

RECORD TYPE 5: SMALL BOX INPUT RECORD
FIELD 1: -4
FIELD 2:
0 single input box
1 two-input box
2 three-input box
FIELD 3: home rung of box
FIELD 4: home column of box
Pass one begins by initializing field one in all 155 records of each rung list to zero. This marks each record as unused. Then, beginning with rung one and working down to rung seven, it examines the matrix cells on each rung from column one to column eleven and processes all existing input elements. Output coils (column 12) are not processed in pass one.

The following sequence describes the pass one process:

1. If a matrix cell contains an element, then the cell is first checked for the presence of a vertical connector on the left. If a vertical connector is present, then a new rung segment begins in this column. To record this, a beginning of rung segment record is added at end of the rung list. If no element is present in the cell, then step one repeats for the cell in the next column to the right.

2. Next, the cell is checked for the type of element present. If the element is a horizontal, a contact, or a large box, then an element record is added at the end of the rung list. If the element is a small box, then a small box input record (record type 5) is added at the end of the rung list. In addition, if the element is a small box and the current rung is the home rung of the small box, an element record for the small box is added at the end of the rung list. If the current rung is not the home rung of the small box, then the algorithm returns to step one for the next cell.

3. Once the element data for the cell is processed, the cell is checked for the presence of a vertical connector on the right. If a vertical connector is present, then the current rung segment ends and an end
of rung segment record is added at the end of the rung list. If one is not present, then the current rung segment continues and a pass through node record is added to the list.

4. This completes the processing of the cell at the current row and column. At this point, the algorithm returns to step one to process the next cell. If the completed cell contained a horizontal or contact, then the next cell is the cell in the column to the right. If the completed cell contained a small box, then the next cell is the cell three columns to the right. Otherwise, if the completed cell contained a large box, then processing continues on the next rung. If the completed cell is in column eleven, then processing continues with the cells on the next rung. Finally, if the completed rung is rung seven, then pass one is complete and pass two begins.

At the completion of pass one, all rungs which contain elements will have rung lists which describe the positions of those elements and define the beginning and end of all rung segments.

Pass one checks for two types of errors. If the reference number of a contact or box is missing, this fact is reported. If an element is missing from any of the cells from column one to twelve on rung one, this is an error and is also reported.
3.5.3 Pass Two

The second pass of the conversion algorithm determines the root rung for each rung segment and adds new information to the rung list of the root rung. To check for errors in the ladder diagram, the root rungs are independently determined at the beginning and end of the rung segments. If the root rungs for both ends of the rung segment are the same, then there is no error. In addition, the segments on the root rung (called root segments) into which the vertical connectors join are also found for error checking purposes.

Pass two processes one rung list at a time beginning with rung seven and working up to rung two. Rung one is not processed since there are no rungs above it with which to join. The process begins by searching for the first occurrence of a beginning of rung segment record (record type 2). The following sequence describes the pass two process:

1. When a beginning of rung segment node record is found, the algorithm stores the node number for this beginning of rung segment node in variable NODE1. If the node is node one, then the root rung is automatically rung one and the root segment is automatically segment one. Otherwise, the root rung is found by checking the vertical connector indicator array NODFLG at the node NODE1 on each successive rung above the current rung until a FALSE value is found. The value FALSE indicates that no vertical connector is present above the rung. Since the algorithm is looking for the last rung that is connected
to the current rung, the first rung which returns a FALSE value is
the root rung. The rung number for the root rung is stored in the
variable ROOT1. Once the root rung is determined, the root segment
must be found. The algorithm does this by searching through the rung
list of the root rung until it finds a rung node record (record type
2) whose node number is the same as NODE1. If such a record is found
the root segment (variable SEG1) becomes the segment number stored
in field four of the record. If no record is found, an error condi-
tion results. In this situation, the error is reported and processing
begins on the next rung at step one.

2. Once the root rung and segment are found for the beginning of the rung
segment, the algorithm searches for the end of the current rung seg-
ment. It does this by checking each record on the current rung. If
no end of rung segment record is found, then an error condition re-
sults and the actions are taken as for the previously described error.
If an end of rung segment record is found, then the root rung and
segment are determined in the same manner as described in step one.
In this step, however, the root values are assigned to the variables
NODE2, ROOT2, and SEG2.

3. After the root rungs and segments are found for each end of the rung
segment, the algorithm compares the root rungs and segments to check
for errors. It is important to note that certain network structures
will cause the root rungs to be different without an error being
present. This step also resolves the difference when those conditions
occur.
First the root rungs are compared. If the root rungs are the same, then the root segments are compared. If the root segments are the same, then the algorithm proceeds to the next step. If they are not the same, then a reverse power flow condition exists in the network. The error is reported and processing continues on the current rung with step 1.

If ROOT1 is greater than ROOT2, then the algorithm will search the ROOT1 rung list until it finds a type 2 record whose node number is the same as NODE2. If the record is not found, then a reverse power flow condition exists in the network. The error is reported and processing continues on the current rung with step 1. If the record is found, then the root segment (SEG2) becomes the segment number stored in field four of the record. Next, the root segments are compared. If the root segments not the same, then a reverse power flow condition exists in the network. The error is reported and processing continues on the current rung with step 1.

If ROOT2 is greater than ROOT1, then a similar process as that described above is followed. In this case, however, the algorithm searches the ROOT2 rung list for a match with NODE1.

4. The next step in this procedure adds two new records to the root rungs to indicate that a parallel rung segment is connected to the root rung. This step is accomplished by searching through the root rung rung list from the end to the beginning and inserting the new records at the appropriate position. Placement of the new records is critical.

CHAPTER 3. METHODOLOGY 54
for the proper operation of pass three. With the records inserted at the correct positions in the lists, each set of two records for a given connecting parallel rung segment will be nested between all sets of two records for each connected parallel rung segment that is below the given segment and connected to the same root rung.

First, an end of parallel segment record (record type 3) is added to the list. This record is placed immediately following the last element record on the root rung segment. Next, a beginning of parallel segment record (record type 4) is added to the list. This record is placed immediately preceding the first element record on the root rung segment. When this step is completed, processing continues at step one for the next rung segment on the current rung. When the end of a rung list is reached, the next rung is processed. When the process is completed on rung two, the algorithm continues with step five.

5. After all rung lists have been processed, the last step of pass two checks for short circuits on all of the rungs. A short circuit is defined as a rung segment which contains only horizontal elements. This step processes one rung at a time beginning with rung one and ending with rung seven. On each rung, the algorithm initializes a short circuit flag (variable SHORTC) to FALSE. It then begins with the first record in the rung list and examines each record until it reaches the end. At the end of a rung list, the next rung is processed. If a node record (record type 2) is found and SHORTC is TRUE, then the node number is stored in NODE2. If a node record (record type 2) is found and SHORTC is FALSE, then the node number is stored
in NODE1. These two variables are used to report the position of short circuits. If a beginning of parallel segment record (record type 4) is found, then SHORTC is set to TRUE and NODE2 is set equal to NODE1. If an element record is found that is not a horizontal element, then SHORTC is set to FALSE. If an end of parallel segment record (record type 3) is found and SHORTC is TRUE, then a short circuit condition exists between the columns stored in NODE1 and NODE2.

At the completion of pass two, all rung lists will contain the necessary information to produce the Boolean code.

3.5.4 Pass Three

The third pass of the conversion algorithm reads the rung lists and generates Boolean code. As each line of Boolean code is generated, it is written to a disk file specified by the user.

This pass employs seven record counters (array REC) to keep track of the last record processed in each rung list. This is necessary because the algorithm switches processing from one rung list to another to operate on parallel rung segments. In addition, pass three uses a twenty-five position stack (array STACK) in order to process parallel rung segments and then return to processing the original rung. When a beginning of parallel rung segment node (record type 4) is found on the rung list, the algorithm pushes the rung number of the parallel segment onto the stack.
When the corresponding end of parallel rung segment node (record type 3) is found, the algorithm swaps the current rung number with the value on the stack. The current rung now becomes that of the parallel rung segment. When processing is complete on the rung segment, the algorithm pops the rung number off the stack and returns to processing the rung list of the original rung at the next record. Another parameter used by pass three is a hold register (variable HOLD). This register allows the algorithm to differentiate between the use of STR and AND during the processing of a rung segment. It also allows differentiation between the use of OR and OR STR at the completion of a parallel rung segment. A start rung flag (variable STRFLG) is employed in pass three to indicate that a new rung segment will start at the next occurrence of an element record.

Pass three begins by initializing the current rung and current record pointers to one. Also, the record counter for rung one is set at one while the record counters for the remaining rungs are initialized to zero. Finally, the hold register and stack pointer are set at zero and the start rung flag is set at TRUE.

There are six steps in the pass three process. The first five are tests that are performed on each rung list record. When the algorithm reads a new record, it begins with the test in step one. If the test fails, then the algorithm performs the test in step two. If that fails, it then tries the next step. This process continues until one of the tests is successfully passed in which case the actions described in that step are performed. At the conclusion of a step, the algorithm reads the next
record on the current rung, and begins the process again with step one. After all records in all rung lists have been read, the algorithm proceeds to step six to process the output coils for the network. The following sequence describes the pass three process:

1. If the rung list record is an element record (type 1), then field two is checked for the type of element. If the element is a horizontal, then the algorithm reads the next rung list record on the current rung and repeats step one. Otherwise, the hold register is checked. If the hold register contains a non-zero value, then the first element for the current rung segment has not been added to the Boolean code. In this case, a STR record for that element added to the Boolean code and the hold register is reset to zero. If the hold register value was greater than zero, then the element is normally open. On the other hand, if the hold register value was less than zero, then the element is normally closed and the NOT operator is added to the Boolean code record. If the hold register equals zero, then no action is taken. After checking the hold register, the algorithm checks the start rung flag. If the start rung flag is TRUE, then the element in the current rung list record is the first element of the rung segment. Therefore, the linked list pointer for that element is placed in the hold register. If the flag is FALSE, then the element is not the first element on the rung segment so field two of the rung list record is checked for the type of element. If the element is a contact, then an AND record is added to the Boolean code for that contact. If the element is a small or large box, then a box record
and the appropriate box data records are added to the Boolean code. At the completion of this step the algorithm reads the next rung list record on the current rung and repeats step one.

2. If the rung list record is a node record (type 2) (this will always be true for the first record of each rung list), then field three is checked for the type of node record. If the record is either a pass through node or a beginning of rung segment node, then the algorithm reads the next rung list record on the current rung and returns to step one. If the record is an end of rung segment node, then the algorithm first checks the hold register. If the hold register is equal to zero, then processing of the rung segment is completed by adding an OR STR record the Boolean code. A non-zero hold register value indicates that the rung segment contains only one element. Therefore, processing of the rung segment is completed by adding an OR record for the element in the hold register to the Boolean code and the hold register is reset to zero. At the completion of this step the algorithm reads the next rung list record on the current rung and returns to step one.

3. If the rung list record is an end of parallel segment record (type 3), then the current rung number is swapped with the value on top of the stack so that the parallel rung segment can be processed. The current rung is now that of the parallel rung segment. The start rung flag is set to TRUE to indicate that a new rung segment has begun. At the completion of this step the algorithm reads the next rung list record on the current rung and returns to step one.
4. If the rung list record is a beginning of parallel segment record (type 4), then the rung number of the parallel rung segment is pushed onto the stack for use later when its end of parallel segment is found. As in step three, the start rung flag is set to TRUE to indicate that a new rung segment has begun. At the completion of this step the algorithm reads the next rung list record on the current rung and returns to step one.

5. If the rung list record is a small box input record (type 5), then field two is checked for the type of small box. If the box is a single input box, then the algorithm reads the next rung list record on the current rung and returns to step one. Otherwise, the algorithm checks field three to see if the current rung is the home rung of the box. If it is the home rung, then the current rung number is pushed onto the stack so that when all input rungs for the box have been processed, the algorithm can continue processing the elements which follow the box. If it is not the home rung, no action is taken. Next, the algorithm checks to see if the current rung is the last input rung for the box. If it is not the last input rung, then current rung becomes the next lower rung and the start rung flag is set to TRUE to indicate the beginning of a new rung segment. If it is the last input rung, then the algorithm pops a rung number off the stack. This becomes the current rung. In either case, processing then continues at step one with the next record on the current rung.

6. After all records on all rung lists have been read, the output coils for a network are processed. The algorithm reads the matrix cell in column twelve on each rung, beginning with rung one and ending at rung
seven. If a cell contains a coil, then the algorithm adds an OUT record to the Boolean code. If not, then it moves to the next cell.

At the completion of step six, pass three is complete. If another network remains to be processed, then the data setup and all three passes are performed on that network. Otherwise the ladder to Boolean conversion is finished for the ladder. If any errors were detected, during the conversion process, then the disk file containing the Boolean code is erased and the total number of errors found is reported. If no errors were found, this fact is reported and the disk file is closed.

The only errors checked in pass three are those associated with output coil reference numbers, and incomplete box data. If coil reference numbers or required box data are missing, the the error is reported and processing continues as if the error had not been detected.

3.5.5 Error Detection

During the ladder to Boolean conversion process, any errors that are detected are reported by printing an error message on the printer. The system will report both the type of error and its location by network, rung, and column. When an error is detected, the algorithm attempts to continues processing as though no error occurred. This allows the entire ladder to be checked for errors. If any errors are found, the disk file containing the Boolean code is erased to prevent the use of incorrect Boolean code files.
Writing to the printer was considered the best choice for this initial system. Writing to the CRT screen would provide more immediate and noticeable results to the user, however, a permanent record would not be available. In addition, a complex means to display the error messages would be required to handle the situation where the number of error messages is too great to fit on one screen. Finally, a means to display the messages while correcting the ladder diagram would be necessary for ease of use. This is beyond the scope of the current system. Writing the error messages to a disk file would be a better solution, but, again, a means to display the messages while correcting the ladder diagram would be necessary.

3.6 BOOLEAN TO LADDER CONVERSION ALGORITHM

The algorithm to convert Boolean code to ladder display data is performed by subroutines BOLDIS and PASSB. It operates on one network at a time and accomplishes its task in two passes. Pass one reads the Boolean code from a disk file and preprocesses data for pass two. Pass two uses the data from pass one and generates the network and element linked lists and other display data. The algorithm assumes that the Boolean code has correct syntax, therefore no error checking is necessary. This section describes how the conversion algorithm works.

The key concept behind this conversion algorithm is that of a nesting of rung segments and Boolean instructions. Every rung segment in a ladder diagram is considered to be an independent level (or nest) of contacts.
or boxes. Whenever a rung segment begins on a ladder diagram, the nesting level increases by one. Whenever two adjacent or parallel rung segments are joined, the nesting level decreases by one. In Boolean code, the beginning of a rung segment is represented by a STR operator. Hence, whenever a STR operator is encountered in the Boolean code, the nesting level increases by one. The joining of two adjacent or parallel rung segments is represented in Boolean code by the AND STR operator and the OR STR operator respectively. So, whenever an AND STR or OR STR operator is encountered in the Boolean code, the nesting level decreases by one. Figure 7 on page 64 shows a simple ladder diagram and the equivalent Boolean code broken down into nesting levels.

Several variables and arrays are shared by both of the conversion algorithm subroutines. Three arrays (FIELD1, FIELD2, FIELD3) store data that is read from the Boolean file. Two additional arrays (BXDATC, BXDATI) store box data that is read from the file. It is important to note at this point that, for this initial programming system, the only small boxes accepted by the algorithm are timers, counters, and ONE SHOT instructions (single input small box). Since only one timer, counter, or large box may appear in a network and since the ONE SHOT requires no data, only one set of box data arrays is required by the algorithm. In the future, when the capability to place more than one single input small box in a network is added to the system, the box data arrays will require one more dimension to handle the additional data. Finally, the array ELENST stores the nesting level of each Boolean instruction and the array ELEINX stores the counter number of each instruction according to its nesting level.
Figure 7. Example of a Ladder Network and its Nesting Levels
counter is described in the section on pass one. Three variables shared by the two subroutines are INDEX, ELLAST, and COUNT. INDEX holds the element list index number of each element and ELLAST is the linked list index number of the last element in a network. The ELLAST value is stored in the network list. COUNT holds the total number of instructions (or elements) that exist in a network.

3.6.1 Pass One

The first pass of this conversion process reads each record from the user specified Boolean file, determines the nesting level of each instruction, and counts the number of instructions in a network. The data variables unique to this first pass include a counter (variable CNTR) to count the number of Boolean instructions read from the disk, a variable to hold the current nesting level (NEST), and an array to store the total number of Boolean instructions on each nesting level (NSTCNT).

The first action of pass one calls the subroutine INITLL which initializes all ladder display data variables and arrays. It will essentially erase any ladder diagram that is currently in the memory of the computer. Next, several element, network, and box counters are initialized to either zero or one. Then, the algorithm reads the first instruction of the first network. The following sequence of events describes the subsequent operation of pass one. Step one is performed once for each network and the remaining steps are executed each time a new instruction is read.
1. The first instruction of a network will contain a STR operator. The algorithm increments the network index number (NTWPTR) and stores the element list index number of the first instruction in the network list. Both the nesting level and the instruction counter are set at one. NSTCNT is set at one for all levels except level one which is initialized to two. Finally, the number of elements on nesting level one (ELENST) is set at one and the counter number of the first element in nesting level one is stored in ELEINX. The iterative process for each network begins with step two.

2. At this step, the instruction counter is incremented by one. The next instruction is read from the Boolean code file and the element data are stored in the data arrays (FIELD1, FIELD2, FIELD3).

3. If the instruction contains a STR operator, then the nesting level (NEST) is incremented by one. Otherwise, no action is taken in this step. Proceed to step ten.

4. If the instruction contains an AND or OR operator or a ONE SHOT box (single input small box) then no action is taken. Proceed to step ten.

5. If the instruction contains a large box element, then the box data records are read from the file and stored in the box arrays. Proceed to step ten.

6. If the instruction contains an AND STR or OR STR operator, then the nesting level (NEST) is decremented by one. Proceed to step ten.

7. If the instruction contains a two-input small box element, then the nesting level is decremented by one. In addition, the box data re-
cords are read from the file and stored in the box arrays. Proceed to step ten.

8. If the instruction contains a three-input small box element, then the nesting level is decremented by two. As in step seven, the box data records are read from the file and stored in the box arrays. Proceed to step ten.

9. If the instruction contains output coil element (OUT operator), then no action is taken.

10. The nesting level for the instruction is stored in array ELENST.

11. If the instruction does not contain a coil element, then the counter for this instruction is stored in array ELEINX, the total number of elements on the current nesting level is incremented by one and the algorithm returns to step two. Otherwise, proceed to step twelve.

12. This step begins the processing of output coil instructions. After the first occurrence of an output coil instruction, all remaining instructions for the network will contain output coils. The first occurrence of an instruction that does not contain an output coil indicates the beginning of a new network. Step 12 increments the instruction counter by one, reads the next instruction from the Boolean code file, and stores the element data in the data arrays (FIELD1, FIELD2, FIELD3). If the instruction contains an output coil, then step twelve repeats. Otherwise, the algorithm proceeds to step thirteen.

13. This step stores the total number of network elements in the variable COUNT and calls pass two. Upon completion of pass two, the algorithm checks to see if the end of the Boolean file has been reached. If
the end of file has been reached, then the Boolean disk file is closed, the final data is added to the display data structure, and the conversion algorithm ends. If the end of file has not been reached, then another network must be processed. The algorithm then repeats the above sequence starting at step one.

3.6.2 Pass Two

Pass two uses the information from pass one to build a network matrix similar to the one developed by the display algorithm. It begins with the first element in the Boolean code and places the information about that element in the proper matrix cell and in the element list. It also accumulates link data for each element in a link matrix (array LINK). This data is later transferred to the element list.

In order to determine the correct location of each network element, four coordinate arrays are used to keep track of where the next element should be positioned. These arrays are: ANDROW, ANDCOL, ORROW, and ORCOL. Each array holds information for a different nesting level. ANDROW and ANDCOL hold the row and column coordinates for the next element (on a given nesting level) that uses an AND operator. Similarly, ORROW and ORCOL hold the row and column coordinates for the next element (on a given nesting level) that uses an OR operator. Each time an instruction is processed, some or all of the coordinate arrays are updated.
At the beginning of pass two, the link matrix is initialized to contain no link data and the number of elements on each nesting level is set at an initial value of one. In addition, the coordinate arrays for nesting level one are set so that the first element on level one will be located at row one, column one and the first element to be ORed with level one will be placed at row two, column one. The procedure then processes each Boolean instruction from the beginning of the network and performs one of the steps below depending upon the type of instruction. Each time an instruction is processed, the nesting level is updated from array ELENST and the total number of elements on the current level (NSTCNT) is incremented by one.

1. If the instruction contains a STR operator, then further checks are made to determine where the element should be located. If the current nesting level is one, then the element is located at the coordinates contained in ANDROW and ANDCOL. ANDCOL is then increased by one. If the nesting level is not one, then the algorithm checks array ELEINX to determine how the rung segment (started by the STR instruction) is terminated. If the rung segment is terminated by an AND STR instruction, then the element is located at the coordinates contained in ANDROW and ANDCOL. If the rung segment is terminated by an OR STR instruction, then the element is located at the coordinates contained in ORROW and ORCOL. In addition, vertical connector information is added to the link data array (LINK) to connect the rung segment with its root rung. If the rung segment is terminated by a two or three input small box, then the element is located at the co-
ordinates contained in ORRON and ORCOL. Additionally, any horizontal elements needed to complete the rung segment and connect it to the box are added to the element list. In each of the last three cases, all four coordinate arrays are updated. In all cases, link data for the element is added to the link array.

2. If the instruction contains an AND operator, then the element is located at the coordinates contained in ANDRON and ANDCOL. Link data is added to the link array and ANDCOL is increased by one.

3. If the instruction contains an OR operator, then the element is located at the coordinates contained in ORRON and ORCOL. For this operator type, further checks must be made to determine what link data should be added to the link array. These checks find out if the element is located in the first column, and whether horizontal elements must be added to the current rung segment before connecting it to its root rung. The horizontal elements would be needed if the root rung segment contained more elements than the current rung segment. Finally, the necessary link data is added to the link array and ORRON is increased by one.

4. If the instruction contains an AND STR operator, then the only action taken is to update ANDCOL and ORRON.

5. If the instruction contains an OR STR operator, then the algorithm determines if horizontal elements must be added to either the current rung segment or the root rung segment before connecting the rung segment to its root rung. The horizontal elements would be added to the rung segment which contained the fewer elements. If both the current rung segment and its root rung segment contain the same number...
of elements, then no horizontal elements are necessary. At the end of this step, link data is added to the link array, and ORROW is updated.

6. If the instruction contains a two- or three-input small box, then the box is located at the coordinates contained in ANDROW and ANDCOL. The box parameters are added to the box data structure, link data is added to the link array, and ANDCOL and ORROW are incremented by three. In addition, if any horizontal elements are needed to connect the box with its input rung segments, then they are added to the element list.

7. If the instruction contains a ONE SHOT (single input small box), then it is located at the coordinates contained in ANDROW and ANDCOL. ANDCOL is then updated.

8. If the instruction contains a drum, then it is located at the coordinates contained in ANDROW and ANDCOL. As in the step for two- and three-input small boxes, the box parameters are added to the box data structure and link data added to the link array. Also, horizontal elements are added to the element list if they are needed. In this step, however, ANDCOL is set at twelve, since no other input elements can follow a large box.

9. If the instruction contains an OUT operator, then the algorithm checks to see if any horizontal elements are needed to complete rung one and connect it with the first output coil. Next, the output coil in the current instruction is located in column 12 at the row contained in ANDROW. Finally, the necessary link data is added to the link array and ANDROW is incremented by one.
When all instructions in a network have been processed, link data contained in the link array is transferred to the element list. Also, the network rungs are counted and the value is stored in the network list. Finally, type 2 elements are added to the element list for those locations in the network matrix where a cell contains a vertical connector on the right (right link type 12) but no element exists. When these operations are completed, then the algorithm returns to pass one to process another network.

3.7 MAN-MACHINE INTERFACE

The man-machine interface was considered to be a very important aspect in the design of this system. Three texts which provided valuable guidance were Gaines and Shaw [4], Ledgard, Singer, and Whiteside [6], and Simpson [16]. Simpson lists twelve principles to follow in designing a system. The system program was developed with these in mind. They are:

1. Define the users.
2. Anticipate the environment in which the system will be used.
3. Give the operator control.
4. Minimize the operator's work.
5. Keep the system simple.
6. Be consistent.
7. Give adequate feedback.
8. Do not overstress the operator's short-term memory.
9. Minimize dependence on recall memory.
10. Help the operator remain oriented.

11. Code information appropriately (or not at all).

12. Follow the prevailing design conventions.

Often, a decision must be made between a menu-driven or command-driven interface when developing interactive software. This decision depends upon the anticipated experience level of the user. The system should provide novices with menus to guide their actions and to quickly familiarize them with the system functions. Experts, however, want to use single key commands or special function keys to execute functions quickly and smoothly. In addition, users at both levels need the ability to back off or escape from an improper step and try again.

It is possible, however, to provide an interface which incorporates the strengths of both the menu-driven and command-driven designs. This has been done in the past with many popular microcomputer software packages, the most notable of which is a dynamic spreadsheet program known as 1-2-3.¹ An attempt was made with the system program to emulate many of interactive features used by 1-2-3, since they have been well-received by a very large population of users and have been accepted by many software development organizations as a de facto standard. This follows principle twelve above.

¹ 1-2-3 is a trademark of Lotus Development Corporation, Cambridge, Massachusetts.

CHAPTER 3. METHODOLOGY 73
The interface provided by 1-2-3 is widely recognized as superior to most because it allows the user to choose between two interaction modes. To perform a function, the user first presses the command key (/). 1-2-3 then displays a list of command words at the top of the screen with one of the command words highlighted in reverse video. Below the list is a one line description of the highlighted command or a list of associated subcommands. Each of the commands may be executed in one of two ways. If the user is familiar with the commands, he or she may simply press the key corresponding to the first letter of the desired command. If the user is not familiar with the commands, he or she may move the reverse video cursor to the desired command with the left or right cursor keys. When the cursor is correctly positioned, pressing the carriage return key will cause the command to be executed.

This interface design is equally useful to users of all experience levels because it provides the speed of key commands and the tutorial guidance of menus. Both methods of operation can be freely mixed together for the benefit of individuals who are very familiar with some command sequences and less experienced with others. Personnel who are using the system after a long period of non-use can easily refresh their memories by using the menu operation until they are once again proficient.

The first important interaction the user has with the system program is with the main menu. The menu, as described earlier in this chapter, is simple and tolerant of errors, yet it provides sufficient information and control to both new and experienced users. The load and save module
screens provide a similar interface. One drawback to the operation of the load and save modules is that they rely heavily on the user's recall memory for entry of a file name. A future improvement to these modules would display a listing of ladder and Boolean files on the current data disk. Also, the name of the current file in memory should be displayed.

The editor interface was more difficult to design than those of the menu, load, and save modules due to its larger number of options and greater complexity. The problem with the large number of options was solved by using a multilevel menu system. This limited the number of available options at any given time to ten which is only slightly more than the optimum $7 \pm 2$ [16]. In addition, this made it possible to effectively use the ten function keys available on the IBM PC keyboard. With few exceptions all editor functions can be performed with the shift, control, and function keys on the left side of the keyboard and with the carriage return and numeric keypad keys on the right side. This minimizes hand movement during editing.

The editor display was kept as free from clutter as possible. The menu and status lines were placed at the bottom of the screen where they would be closer to the keyboard. This would tend to minimize eye movement for new users who must consult the editor menu before keying a function. Although status information might be better placed at the top of the screen where it would be at eye level, it was decided that separating the menu and status lines would tend to disorganize the screen.
It was the initial objective of this thesis to develop a ladder diagram editor with ability to scroll through a ladder rung by rung rather than network by network. The former approach would give the user maximum flexibility in moving around a ladder diagram just as a full screen text editor does for editing text files. However, it was determined that a rung by rung scrolling editor would require a much larger ladder data structure. This requirement would cause the system program to be too large for use on a personal computer with only 256K of memory. There were other conflicts with this type of editor. For example, in the current system, different element types occupy varying amounts of space on the editor screen. A small box covers three rows and three columns whereas a contact only occupies one of each. In certain situations, scrolling one rung could cause only portions of small or large boxes to be drawn on the screen. This was judged to be unacceptable. The development of a more sophisticated display algorithm and writing the system program completely in assembly language may solve these problems. However, for the purposes of this thesis, the network by network scrolling scheme proved to be sufficient.

Many of the problems and questions that arose during the design of this system's man-machine interface could not be adequately resolved because they would involve experimentation beyond the scope of the thesis. For example, to select an X, Y, or C type contact, the user must press a function key corresponding to the contact type (F5, F7, or F9). It was decided to use the function keys for this task rather than the X, Y and C keys because it reduced the amount of hand motion during programming.
There is a disadvantage in using the function keys, though. The operator must either memorize the key correspondence or refer to the editor prompt to select the desired contact type. To properly resolve the issue on which keys to use, tests should be conducted which indicate whether reducing hand motion or minimizing dependence on recall memory is the more important criterion.

In addition many characteristics of this system made it difficult to incorporate conventions that are standard with more common types of software such as text editors and spreadsheets. For these reasons, many improvements can be made to this system to increase its power and ease of use. Some of these enhancements will be discussed in chapter five.
4.0 CHAPTER 4. DEVELOPMENT OF A UNIVERSAL PROGRAMMING SYSTEM

4.1 EXPANSION TO COMPLETE TI MODEL 530 INSTRUCTION SET

With the Phase One system completed, it was possible to assess the feasibility of expanding it to allow programming of the entire TI Model 530 instruction set. Then, a strategy could be developed to achieve that objective. There are two problems that must be overcome in order to expand the system to the full instruction set. The first is programming single input small boxes. The second is expanding the data structure to accommodate the additional data required by most of the advanced instructions (small and large boxes).

Most of the box instructions were not included in the initial system's instruction set because they provide very advanced functions beyond the scope of a basic programming system. This exclusion was beneficial because it avoided the problem of box placement. All two- and three-input small boxes in the Model 530 instruction set must be located on rung one and in either column one or column two. (This requirement may have been established by Texas Instruments in order to reduce the complexity of its own programming devices rather than because of a limitation in the controller itself.) Single input small boxes, which comprise a majority of the box instructions, may be placed anywhere on the input side of the network display area. This freedom of placement would add further complexity to several of the editor functions and the conversion algorithms.
For example, when adding a small box to a network, the editor must check to see if any ladder elements will be written over. If this is true, then the editor must remove the element or elements from the ladder data structure. This involves not only removing the element from the element list, but also making changes to the network and type reference lists if necessary and updating the element list pointers in the display matrix cells. In addition, the editor must remove any vertical connections that were associated with the deleted elements. A further check which must be made with single input small boxes is that of ensuring that a small box will not be placed over a portion of another small box. All of this is done now by the current system, but the problem is made less difficult by the fact that all available small boxes may be placed at only two locations.

It is evident, then, that introducing the remaining small box instructions will place greater demands on the system to prevent operator errors. However, the ladder diagram data structure was designed with these future problems in mind. Therefore, the expansion, in terms of the above capabilities, should not be impossible.

The remaining elements in the instruction set employ a very wide variety of data due to their diverse functions. However, the processing of that data by both the editor and the conversion algorithms is facilitated by a common format shared by all of the instructions. All small box instructions, with four exceptions, contain either three or four data values, including the box reference number. The exceptions include the One
Shot (one value), the Counter and Timer (two values), and the Compare (five values). This consistency among the instructions would allow the system program to store data on all the various instructions in one common array with little wasted storage space. Thus, only one data access procedure would be required. Since only a finite number of small box instructions can be programmed into the fixed memory size of the Model 530, the size of the array should be manageable.

A problem arises, though, with the addition of large box instructions. The only large box provided by the initial system, the Drum, requires the least amount of storage. To store data for every possible drum allowed by the Model 530 requires 3,000 bytes of memory. The other three types of large boxes, the Event Drum, the Scan Matrix Compare, and the Indexed Matrix Compare, each need 3,960, 17,340, and 16,830 bytes respectively. It is impossible to program all 255 of either Matrix Compare instruction using the limited memory of the Model 530, however. For either of those two instructions, then, a more realistic memory requirement is approximately 8,000 bytes. That number is still conservative, since it would entail programming the controller with nothing but the matrix compare Instructions and their accompanying contact and coil.

To expand the system with respect to memory needs, then, it will be necessary to enlarge the small box data array, provide an array for the two types of drums and an array for the two types of matrix compare instructions. Since a large box data array already exists for Drum instructions, the second requirement can be met by enlarging that array.
In order to determine the feasibility of expanding the initial system program to programmable controllers produced by other manufacturers, it was necessary to review their instruction sets. Three manufacturers were chosen based upon their combined share of the American programmable controller market and the availability of reference materials from them. They were Allen-Bradley Company, and General Electric Company, and Gould Incorporated. These three companies with Texas Instruments and a fifth company, Siemens-Allis Automation Incorporated, comprise well over fifty percent of the market. Reference material was obtained for Siemens-Allis controllers, however, it did not furnish sufficient information on the controller instruction sets.

From the three companies, programmable controller models which were most popular and which contained the most diverse instruction sets were selected. Instruction set information (references [23] through [32]) was obtained for the programmable controllers listed below.

- Allen-Bradley — PLC-2/05, PLC-2/20, PLC-2/30
- General Electric — Series 6
- Gould — Modicon 884, Modicon 584

A summary of the instruction sets for each of these controllers is contained in Appendices C, D, and E. A summary of the TI Model 530 instruction set is contained in Appendix B.
The instruction sets for all of the controllers can be divided into six functional groups. They are:

- Relay Logic,
- Timing and Counting,
- Arithmetic,
- Data Manipulation,
- Data Transfer, and
- Program Control.

The first two groups are considered to be the most basic and are available on almost every programmable controller made [1]. The last four comprise the more advanced instructions.

4.2.1 Comparison of Instruction Sets

The review of instruction sets indicated that all of the programmable controllers provide instructions from each group. The differences exist in the number of instructions from each group and the manner in which they are programmed. In this section, a comparison of the programmable controllers will be made on the basis of each group of instructions. One reference which facilitated this comparison was Bryan and Jones [1] who list and describe a comprehensive superset of instructions.
4.2.1.1 Relay Logic Group

In the relay logic group, the common instructions among the controllers are identical. Any differences between the controllers existed in the number of different instructions each provided. There are eight instructions in the superset for this group:

1. Normally Open Contact,
2. Normally Closed Contact,
3. Normally Deenergized Coil,
4. Normally Energized Coil,
5. Latch Coil,
6. Unlatch Coil,
7. Positive Transitional Contact (One Shot), and
8. Negative Transitional Contact.

All of the controllers that were investigated provide the first three instructions in their instruction sets, but none of them include all eight. Only the TI Model 530 has normally energized coils. The most notable difference among the controllers was in the implementation of One Shot instructions. In Gould controllers, a Positive Transitional Contact is used which passes power whenever its reference data bit changes from 0 to 1. The TI Model 530 controller uses a One Shot box instruction which produces a TRUE value on its output rung (essentially passing power) when its input rung changes from a FALSE to a TRUE value. The GE controller uses a One Shot coil which turns on when its input rung changes from FALSE
to TRUE. Thus, the GE One Shot can directly control an output signal or control relay, whereas the other two types of One Shot instructions can only indirectly affect output coils or control relays. The end result would be the same, since equivalent networks can be designed with any of the three controllers.

4.2.1.2 Timer and Counter Group

More diversity existed in the instructions from this group and the following groups than in those of the Relay Logic Group. The superset of instructions for this group includes seven instructions:

1. Timer On,
2. Timer Off,
3. Retentive Timer On,
4. Retentive Timer Reset,
5. Up Counter,
6. Down Counter, and
7. Counter Reset.

The differences among controllers in this group exist in the way the instructions are programmed. Allen-Bradley controllers provide all seven superset instructions as they are listed above. All of the other controllers employ functional boxes which combine some of the superset instructions into one instruction. For example, the TI Model 530 provides a Timer, an Up Counter, and an Up/Down Counter. The timer is a retentive
timer which has two input rungs. An enable/reset rung resets the timer to a preset value when the rung is FALSE. When the enable rung is TRUE and a start/stop rung is TRUE the timer will time. When the enable rung is TRUE and the start/stop rung is FALSE, then the timer will retain its accumulated value, but will not time.

The TI Model 530 controller permits timer intervals in 0.1 and 0.001 second increments. The timers for all of the other controllers use 1.0, 0.1 and 0.01 second increments. Maximum timer preset values varied among the controllers from 999 to 65,535.

4.2.1.3 Arithmetic Group

The Arithmetic Group consists of five basic instructions:

1. Add,
2. Subtract,
3. Multiply,
4. Divide, and
5. Square Root.

All of the controllers provide the first four instructions. Some, however, enhance them by providing double precision options. The basic Add and Subtract instructions operate on values stored in sixteen bit registers, whereas their double precision counterparts operate on values stored in two consecutive registers. The Multiply, Divide, and Square
Root functions also use two consecutive registers for their operands or results.

Allen-Bradley controllers, which implement the instructions as coils, set error bits on underflow or overflow conditions. The other controllers program the instructions with function boxes that have one or more output rungs to turn on output coils in the event of an error.

Another significant difference between the controllers is caused by the way they each store values. Allen-Bradley, for example, only allows signed three digit BCD numbers to be stored in a sixteen bit register. The other four register bits are used for various indications or not at all. Thus, the range of value that can be stored in a register is -999 to 999. Gould uses all sixteen bits in a BCD format, so it can store unsigned values in one register up to a maximum of 9,999. GE and Texas Instruments both store signed binary numbers which gives them a range from -32,768 to 32,767.

4.2.1.4 Data Manipulation Group

The widest variety of instructions among the controllers exists in this group and the Data Transfer Group in the next section. As in the Timer and Counter Group, some instructions are combined into one. None of the controllers provide all of the instructions listed below.
1. Compare Equal,
2. Compare Greater Than,
3. Compare Less Than,
4. Conversion instructions,
5. Logical AND,
6. Logical OR,
7. Logical XOR,
8. Logical NOT,
9. Logical Compare,
10. Bit Set,
11. Bit Clear,
12. Bit Check,
13. Shift Left
14. Shift Right,
15. Rotate Left,
16. Rotate Right,
17. Drum (Sequencer), and

The first three instructions perform mathematical compares on values stored in two registers. The results of those functions either turn on a coil or cause an output rung to pass power depending on the controller. The conversion instructions convert binary values to BCD for output display devices and convert BCD input into binary values.

The remaining instructions in this group manipulate the bits in one or more registers. The number of bits or registers operated by a given instruction depends upon the controller. For example, the TI Model 530 provides a logical AND instruction which ANDs two sixteen bit registers and places the result in a third register. To perform the operation on several consecutive registers, the programmer must write a ladder routine to step through the registers. Two of the Gould controllers, however, have a logical AND operation which automatically AND a specified number
of consecutive registers from one to one hundred. Another significant
difference between these two controllers is that the TI Model 530 places
the results of its logical operations in a third register, whereas the
Gould controllers destroy the contents of the second operand register.

4.2.1.5 Data Transfer Group

The instructions in this group all move data from one or more memory lo-
cations to other memory locations. As in the last group, the instructions
differed enough from controller to controller that none provided all of
the instructions in the following list:

1. Get,
2. Put,
3. Register to Table Move,
4. Table to Register Move,
5. Table to Table Move,
6. Block Move,
7. First In Move,
8. First Out Move,
9. Last Out Move,
10. Table Search, and
11. Table Sort.

The first two instructions are used to transfer the contents of one reg-
ister to another. The GE controller even provides two instructions to
move either half of a register (one byte). With just the Get and Put instructions, it would be possible to perform any of the other operations in the list. The other instructions simply provide a higher level of control.

4.2.1.6 Program Control Group

This last group consists of instructions which alter the controller's normal program scan. The instructions are:

1. Immediate Input,
2. Immediate Output,
3. Master Control Relay,
4. Zone Control Last,
5. Skip and Deenergize,
6. Skip and Retain,
7. Jump,
8. Label,
9. Jump to Subroutine, and
10. Return.

All of the instructions in this group perform simple operations which do not vary from controller to controller. The only differences found between controllers in this group is in the number of instructions the controller provides.
4.2.2 Summary

In order to develop a universal programming/documentation system, it will be necessary to complete three major tasks. First, a universal instruction set must be designed. The study of programmable controller instruction sets revealed that each controller possesses a subset of instructions from the lists shown in the previous sections. Some instructions can be combined into one, more powerful instruction without restricting flexibility. Therefore, the universal set would have to contain sufficient instructions to adequately represent all of the instructions provided by the target controllers.

The second task is to develop a consistent format with which to graphically represent the universal set in a ladder diagram display. It was found that some controllers used output coils to represent functions and others used function boxes. Both formats, however, could be used to design equivalent control programs. Thus, the format should be chosen by how well it displays information, how easy it is to use, how flexible it is in converting to a target controller, and which formats predominate the programmable controller market.

The final task will be to create interface programs which can convert the universal ladder diagrams and download the correct information to the target controllers. In the case of the TI Model 530 controller, the initial system program would convert ladder diagrams to the Boolean language and download it through a network communications line.

For relay logic
instructions, this language is the same for all controllers. If it is found that all controllers use the same Boolean format for advanced functions, then the Boolean language can be output by the interface programs and downloaded in a similar manner. If this is not the case, then it will be necessary for the interface programs to download data directly to the controllers' memory.
5.0 CHAPTER 5. PROGRAM ANALYSIS

As with any piece of software, the act of creating the initial system program was a learning experience which pointed out areas for future improvement. This chapter will discuss some improvements in the areas of program functions and operator interface. The intention in suggesting these improvements is to further enhance the capabilities and acceptability of this programming/documentation system. Their incorporation will provide a more reliable, useful, friendly system.

5.1 PROGRAM FUNCTIONS

1. Ladder Printout Function
One of the system objectives which was not implemented was the ladder diagram printout function. This capability is greatly needed because it provides a permanent record of a program and aids in the process of debugging a program. Currently, to obtain a printed copy of a program, it is necessary to display each network on the screen and use the IBM PC Print Screen key. One further extension to this function would be to provide the ability to read a Boolean file and print out the Boolean code.

2. Cross Reference Function
Another system objective not implemented was the ladder diagram cross reference facility, which is provided by most programmable controller documentation systems. It is another valuable aid to program debugging. This feature would allow the creation of element lists organized by ref-
ference number or element type and would provide the location of the ele-
ments by network, rung, row, and column. This function should permit
cross references to be displayed on the screen or printed out on a
printer.

3. Search/Replace Functions

In addition to the current means of moving around a ladder program by
network, programming efficiency could be enhanced by a search function.
The operator should be able to locate and display any element by entering
its reference number, or any rung or network. An associated replace
function would also be helpful. With it, the programmer could perform
single or global replacements of any ladder element.

4. Insert Row/Column and Delete Row/Column

Four related functions which are needed in the Editor module are Insert
Row, Insert Column, Delete Row, and Delete Column. They would be par-
ticularly necessary if the ability to scroll by rung is added to the
system. Insert Row would simply move all elements in a specified row on
the screen down one row and open up a new row for the placement of ele-
ments. Insert Column would work similarly for a column of elements.
Delete Row and Delete Column would enhance the Delete Element function
by deleting all elements on a row or in a column. The capability to add
these functions to the editor menu system already exists through the use
of the Delete and Insert keys.

CHAPTER 5. PROGRAM ANALYSIS 95
5.2 OPERATOR INTERFACE

1. Editor Scrolling
As discussed in the Chapter 3 section on man-machine interface, it was originally planned that the Editor module would permit scrolling through a ladder diagram rung by rung. This is still considered to be the best way to display ladder diagrams since it permits the user to observe more than one rung at a time. This ability would greatly aid both programming and debugging tasks.

2. Help Screens
Most good software systems provide on-line help facilities which provide brief explanations of program operation. One effective means to accomplish this in the system program would be to assign one specific key or key combination as a help key and establish a set of pop up help screens for each program function. When the help key is pressed, a menu of help screens would appear on the screen. The operator could select a help screen by moving a highlighted cursor to the appropriate entry and press the enter key. This would cause the specific help screen to appear. When finished with the help screen or help menu, the operator would press the help key to exit the help function and return to the main program.

3. Network Comments and Element Labels
Two means to make programs easier to understand would be to add network comments and element labels. Network comments could be most effectively implemented by creating a pop up screen facility. This would prevent wasting valuable screen space. In order to add, edit, or display the comments for the current network the user could press a designated comment
key. This would cause a comment area to appear on the screen. To remove the comment area, the same key could be pressed. As long as the comment area is on the screen, the user could enter or edit any desired comments.

4. Error Handling

Complete error handling is not available in the Editor module as it is in the Main Menu, Save, and Load modules. Currently, if the operator presses an invalid key, for example, the program will sound a beep tone, but will not display an error message. To correct this deficiency, it will be necessary to write an error message subroutine similar to that used by the Main Menu, Save, and Load Modules.
The work performed for this thesis involved the development and implementation of a programming/documentation system for a Texas Instruments Model 530 programmable controller. The system was designed to run on a host IBM personal computer, IBM portable personal computer, or IBM personal computer XT. It allows an operator to create and edit programmable controller programs using a ladder diagram language format and then convert those programs into the equivalent Boolean code. Programs may be saved on a diskette in either a ladder diagram or Boolean code format and later retrieved for further editing.

The programming/documentation system was primarily developed as a prototype for a universal programming/documentation system capable of programming controllers manufactured by several different vendors. The system's development created a knowledge base on which to build more sophisticated algorithms for the purposes of storing ladder diagram information, converting ladder diagram information to Boolean code, and converting Boolean code to ladder diagram information. It also provided insights into the feasibility of creating a truly universal programming/documentation system. Upon completion of the initial system, a review of selected programmable controller instruction sets was performed. This review indicated that all instructions for programmable controllers can be categorized into seven distinct functional groups. Therefore, a universal programming/documentation system would have to
provide instructions which represent all of the functions available in each of the groups.

In conclusion, it is possible to develop a universal programming/documentation system based on the initial system developed in this thesis. The endeavor will require that a universal instruction set be created from the instruction sets of the various target controllers. There may be some limitations to such an instruction set that will prevent the universal programming/documentation system from taking full advantage of some controllers with unique functions. This must be assumed when first developing a general programming system. However, with time and the cooperation of programmable controller manufacturers, a standard programming language could be developed which would eliminate such inconsistencies.

The development of a universal system will also require modifications to the initial system, particularly the editor, to implement the universal instruction set. As already stated, the initial system was designed to facilitate those changes. Finally, interfaces to the target controllers must be developed. This task will be the most difficult as it will also require the cooperation of programmable controller manufacturers, but it is not impossible.
LIST OF REFERENCES


19. Literature on programmable controller documentation packages. Taylor Industrial Software, 12204-106 Avenue, Edmonton, Canada T5N 3Z1.


APPENDIX A. SYSTEM PROGRAM INFORMATION

PROGRAMMING/DOCUMENTATION SYSTEM DATA STRUCTURE

ELEMENT LIST

ELHEAD - pointer to first entry in element list
ELFREE - pointer to first unused record in element list

ELMNTC - Element List Character Array

FIELD | DATA
----- |-----
1     | element type code
2     | row coordinate
3     | column coordinate
4     | link data

ELMNTI - Element List Integer Array

FIELD | DATA
----- |-----
1     | backward pointer
2     | forward pointer
3     | element reference number or pointer to box data list

NETWORK LIST

NWHEAD - pointer to first entry in network list
NWTAIL - pointer to last entry in network list
NWFREE - pointer to first unused record in network list

NETWRK - Network List Array

FIELD | DATA
----- |-----
1     | element list pointer for first element in network
2     | element list pointer for last element in network
3     | number of rungs in network
4     | backward pointer
5     | forward pointer
ELEMENT TYPE CODES

00 null element
01 horizontal element
02 vertical connector element

Contacts:

- 03 X normally open external input contact
- 04 Y normally open external output contact
- 05 C normally open control relay contact

- 06 X normally closed external input contact
- 07 Y normally closed external output contact
- 08 C normally closed control relay contact

Coils:

- 09 (not used)
- 10 Y normally deenergized external output coil
- 11 C normally deenergized control relay coil

- 12 (not used)
- 13 Y normally energized coil external output coil
- 14 C normally energized coil control relay coil

15 JMP jump coil
16 JMP(E) end jump coil
17 MCR master control relay coil
18 MCR(E) end master control relay coil
19 END(U) unconditional end coil
20 END(C) conditional end coil
Small single-input boxes:

- O/S 0/S one shot box
- ADD add box
- SUB subtract box
- DIV divide box
- MULT multiply box
- SQRT square root box
- CMP compare box
- WAND word AND box
- WOR word OR box
- WXOR word XOR box
- WROT word rotate right box
- MHIR move word to image register box
- MIRW move image register to word box
- MOVW move word box
- LDC load data constant box
- BITP bit pick box
- BITC bit clear box
- BITS bit set box
- CBD convert binary to BCD box
- CDB convert BCD to binary box

Small two-input boxes:

- CTR counter box
- TMR timer box
- MWFT move word from table box
- MWTT move word to table box

Small three-input boxes:

- SHRB bit shift register box
- SHRW word shift register box
- UDC up/down counter box
Large boxes:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>SMC scan matrix compare box</td>
</tr>
<tr>
<td>LARGE BOX</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>51</td>
</tr>
</tbody>
</table>
LINK TYPES FOR INPUT LADDER ELEMENTS

LEFT LINK TYPES:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>no link</td>
</tr>
<tr>
<td>03</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Appendix A. SYSTEM PROGRAM INFORMATION
### RIGHT LINK TYPES:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE 00</td>
<td>no link</td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE 03</td>
<td></td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE 05</td>
<td></td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE 06</td>
<td></td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE 07</td>
<td></td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE 09</td>
<td></td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TYPE 10</td>
<td></td>
<td><img src="image" alt="Example" /></td>
</tr>
<tr>
<td>(10)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LINK TYPES FOR OUTPUT LADDER ELEMENTS

**LINK TYPES:**

**TYPE 03**

(48)

---

**TYPE 06**

(96)

---

**TYPE 11**

(176)

---

**TYPE 14**

(224)

---

**EXAMPLES:**

---

---

---

---
### SUMMARY OF LINK TYPES

#### LINK COMPONENTS

<table>
<thead>
<tr>
<th>LEFT LINK COMPONENT</th>
<th>BIT VALUE</th>
<th>RIGHT LINK COMPONENT</th>
<th>BIT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>128</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

#### LINK TYPES

<table>
<thead>
<tr>
<th>LEFT LINK TYPE</th>
<th>BIT VALUE</th>
<th>RIGHT LINK TYPE</th>
<th>BIT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>48</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>80</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>96</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>112</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>144</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>160</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>176</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>192</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>208</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>14</td>
<td>224</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>15</td>
<td>240</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

Appendix A. SYSTEM PROGRAM INFORMATION 110
APPENDIX B. TEXAS INSTRUMENTS MODEL 530 INSTRUCTION SET

1. Relay Logic Instructions

Contacts

a. Normally Open Contact

b. Normally Closed Contact

c. One Shot O/S

Coils

a. Normally Energized Coil

b. Normally Deenergized Coil
2. Timing and Counting Instructions

Timers

a. Timer

TMR timer times when input rung and enable rung are TRUE; timer resets to zero when enable rung is FALSE; output rung passes power when accumulated time reaches preset value;

Timer increments: 0.1 second, 0.001 second
Timer range: 0-32,767

Counters

a. Counter

CTR increments accumulated count each time input rung changes from FALSE to TRUE when enable rung is TRUE; count value resets to zero when enable rung is FALSE; output rung passes power when accumulated count equals preset value

b. Up/Down Counter

UDC when enable rung is TRUE, increments accumulated count if UP rung changes from FALSE to TRUE and decrements count if DOWN rung changes from FALSE to TRUE; count value resets to zero when enable rung is FALSE; output rung passes power when count reaches preset value or zero; a specified coil goes active when count value is zero

Counter range: 0-32,767
3. Arithmetic Instructions

a. Add  
ADD adds the signed values in two memory words and places the result in a third word when input rung is TRUE; output rung passes power if no error occurs

b. Subtract  
SUB subtracts the signed value in one memory word from that in another word and places the result in a third word when input rung is TRUE; output rung passes power if no error occurs

c. Multiply  
MULT multiplies the signed values in two memory words and places the result in two consecutive memory words when input rung is TRUE; output rung passes power if no error occurs

d. Divide  
DIV divides the signed value in two memory words by the value in a third word and places the quotient and remainder in two consecutive memory words when input rung is TRUE; output rung passes power if no error occurs

e. Square Root  
SQRT takes the square root of a positive signed value in two consecutive memory words and places the result in a third word; output rung passes power if no error occurs
4. Data Manipulation Instructions

a. Compare CMP compares the values in two memory words when input rung is TRUE; output rung passes power if the values are equal; causes a specified coil to go active if first value is less than the second; causes another specified coil to go active if first value is greater than the second.

b. Indexed Matrix IMC compares the bit pattern of a specified step in a data matrix with up to 15 I/O points when input rung is TRUE; operation uses the step value in a specified memory word if enable rung is TRUE; output rung passes power if an exact bit match occurs.

c. Scan Matrix SMC compares the bit pattern of up to 16 patterns in a data matrix with up to 15 I/O points when input rung is TRUE; places step value of pattern in a specified memory word and output rung passes power if a match is found.

d. Convert Binary to Decimal CBD converts the value in a specified memory word from binary to BCD and places the result in two consecutive words when input rung is TRUE; output runs power when input rung is TRUE.

e. Convert BCD to Binary CDB converts a BCD value in a specified memory word to binary and places the result in another word when input rung is TRUE; output rung passes power when input rung is TRUE.

f. Word AND AND logically ANDs two memory words and places the result in a third word when input rung is TRUE; output rung passes power when input rung is TRUE unless result is zero.

g. Word OR OR logically ORs two memory words and places the result in a third word when input rung is TRUE; output rung passes power when input rung is TRUE unless result is zero.
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>h. Word XOR</td>
<td>XOR performs exclusive OR on two memory words and places result in a third word when input rung is TRUE; output rung passes power when input rung is TRUE unless result is zero.</td>
</tr>
<tr>
<td>i. Bit Set</td>
<td>BITS sets a specified bit in a specified memory word to 1 when input rung is TRUE; output rung passes power if input rung is TRUE.</td>
</tr>
<tr>
<td>j. Bit Clear</td>
<td>BITC sets a specified bit in a specified memory word to 0 when input rung is TRUE; output rung passes power if input rung is TRUE.</td>
</tr>
<tr>
<td>k. Bit Pick</td>
<td>BITP examines the status of a specified bit in a specified memory word when input rung is TRUE; output rung passes power if bit is 1.</td>
</tr>
<tr>
<td>l. Bit Shift</td>
<td>SHRB shifts up to 16 bits beginning at a specified memory word when bottom input rung is TRUE and top input rung changes from FALSE to TRUE; status of middle input rung (0 for FALSE, 1 for TRUE) is shifted into open bit position; output rung passes power if bit shifted out of register is 1.</td>
</tr>
<tr>
<td>m. Word Rotate</td>
<td>WROT rotates 4-bit segments to the right in a specified memory word when input rung is TRUE; output rung passes power when input rung is TRUE.</td>
</tr>
<tr>
<td>n. Drum</td>
<td>DRUM turns on or off up to 15 coils according to up to 16 step masks when enable rung and input rung are TRUE; drum remains on a step for a specified period of time; step resets to a specified reset step when enable rung is FALSE; output rung passes power when last step completes.</td>
</tr>
<tr>
<td>o. Event Drum</td>
<td>EVENT DRUM operates similarly to DRUM except EVENT DRUM increments steps when specified event contacts close or when step times out.</td>
</tr>
</tbody>
</table>
### 5. Data Transfer Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a. Load Data Constant</strong></td>
<td>LDC</td>
</tr>
<tr>
<td><strong>b. Move Image Register to Word</strong></td>
<td>MIRW</td>
</tr>
<tr>
<td><strong>c. Move Word to Image Register</strong></td>
<td>MRIW</td>
</tr>
<tr>
<td><strong>d. Move Word to Word</strong></td>
<td>MOVW</td>
</tr>
<tr>
<td><strong>e. Move Word from Table</strong></td>
<td>MWFT</td>
</tr>
<tr>
<td><strong>f. Move Word to Table</strong></td>
<td>MWTT</td>
</tr>
<tr>
<td><strong>g. Word Shift Register</strong></td>
<td>SHRW</td>
</tr>
</tbody>
</table>
6. Program Control Instructions

a. Master Control Relay

MCR when MCR coil is inactive, all coils within MCR field of control are turned off; when MCR is active, all coils within field of control operate according to internal logic.

b. Jump

JMP when JMP coil is inactive, all coils within JMP field of control remain in their last state; when JMP is active all coils within field of control operate according to internal logic.

c. JMP End or MCR End

-(E)- ends influence field of JMP and MCR instructions

d. Unconditional End

-(U)- ends program scan

e. Conditional End

-(C)- ends program scan if input rung is TRUE
## INSTRUCTION SET FOR TEXAS INSTRUMENTS
### MODEL 530 CONTROLLER

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>MNEMONIC</th>
<th>NUMBER ALLOWED</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTACT</td>
<td>X, Y, or C</td>
<td>1023 1</td>
</tr>
<tr>
<td>COILS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coil</td>
<td>Y</td>
<td>1023 1</td>
</tr>
<tr>
<td>Control Relay</td>
<td>C</td>
<td>511</td>
</tr>
<tr>
<td>Jump</td>
<td>JMP</td>
<td>8</td>
</tr>
<tr>
<td>End Jump</td>
<td>JMP (E)</td>
<td>8</td>
</tr>
<tr>
<td>Master Control Relay</td>
<td>MCR</td>
<td>8</td>
</tr>
<tr>
<td>End Master Control Relay</td>
<td>MCR (E)</td>
<td>8</td>
</tr>
<tr>
<td>End (unconditional)</td>
<td>END (U)</td>
<td>× 2</td>
</tr>
<tr>
<td>End (conditional)</td>
<td>END (C)</td>
<td>× 2</td>
</tr>
<tr>
<td>COUNTER-TIMER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter</td>
<td>CTR</td>
<td>128 3</td>
</tr>
<tr>
<td>Timer</td>
<td>TMR</td>
<td>128 3</td>
</tr>
<tr>
<td>Up-Down Counter</td>
<td>UDC</td>
<td>128 3</td>
</tr>
<tr>
<td>One-Shot</td>
<td>O/S</td>
<td>128</td>
</tr>
<tr>
<td>MATH</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>ADD</td>
<td>256</td>
</tr>
<tr>
<td>Subtract</td>
<td>SUB</td>
<td>256</td>
</tr>
<tr>
<td>Multiply</td>
<td>MULT</td>
<td>256</td>
</tr>
<tr>
<td>Divide</td>
<td>DIV</td>
<td>256</td>
</tr>
<tr>
<td>Compare</td>
<td>CMP</td>
<td>256</td>
</tr>
<tr>
<td>Square Root</td>
<td>SQRT</td>
<td>256</td>
</tr>
<tr>
<td>DRUM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time-based Drum</td>
<td>DRUM</td>
<td>30 4</td>
</tr>
<tr>
<td>Event &amp; Time Based Drum</td>
<td>EVENT DRUM</td>
<td>30 4</td>
</tr>
<tr>
<td>MATRIX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Indexed Matrix Compare</td>
<td>IMC</td>
<td>256</td>
</tr>
<tr>
<td>Scanned Matrix Compare</td>
<td>SMC</td>
<td>256</td>
</tr>
</tbody>
</table>
### BIT

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit Clear</td>
<td>BITC</td>
<td>256</td>
</tr>
<tr>
<td>Bit Pick</td>
<td>BITP</td>
<td>256</td>
</tr>
<tr>
<td>Bit Set</td>
<td>BITS</td>
<td>256</td>
</tr>
<tr>
<td>Bit Shift Register</td>
<td>SHRB</td>
<td>30</td>
</tr>
</tbody>
</table>

### WORD

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convert Binary to BCD</td>
<td>CBD</td>
<td>256</td>
</tr>
<tr>
<td>Convert BCD to Binary</td>
<td>CDB</td>
<td>256</td>
</tr>
<tr>
<td>Word Rotate</td>
<td>WOR</td>
<td>256</td>
</tr>
<tr>
<td>Word AND</td>
<td>WAND</td>
<td>256</td>
</tr>
<tr>
<td>Word OR</td>
<td>WOR</td>
<td>256</td>
</tr>
<tr>
<td>Word XOR</td>
<td>WXOR</td>
<td>256</td>
</tr>
<tr>
<td>Word Shift Register</td>
<td>SHRW</td>
<td>30</td>
</tr>
</tbody>
</table>

### MOVE

<table>
<thead>
<tr>
<th>Operation</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Data Constant</td>
<td>LDC</td>
<td>256</td>
</tr>
<tr>
<td>Move Image Register to Word</td>
<td>MIRW</td>
<td>256</td>
</tr>
<tr>
<td>Move Word to Image Register</td>
<td>MWIR</td>
<td>256</td>
</tr>
<tr>
<td>Move Word</td>
<td>MOVW</td>
<td>256</td>
</tr>
<tr>
<td>Move Word from Table</td>
<td>MWFT</td>
<td>30</td>
</tr>
<tr>
<td>Move Word to Table</td>
<td>MWTT</td>
<td>30</td>
</tr>
</tbody>
</table>

Total Instructions - 39

**NOTES:**

1. The total number of contacts and coils may not exceed 1023.
2. The number allowed is limited by program memory.
3. The total number of timers, counters, and up-down counters may not exceed 128.
4. The total number of drums and event drums may not exceed 30.
5. The total number of SHRW and WOR instructions may not exceed 30.
6. The total number of MWFT and MWTT instructions may not exceed 30.
APPENDIX C. ALLEN-BRADLEY INSTRUCTION SETS

ALLEN-BRADLEY PLC-2/20 INSTRUCTION SET

1. Relay Logic Instructions

Contacts
a. Examine On
   contact is open (open circuit) when data table bit is 0 and closed (short circuit when data table bit is 1.

b. Examine Off
   contact is closed (short circuit) when data table bit is 0 and open (open circuit) when data table bit is 1.

Coils
a. Output Energize
   sets data table bit to 0 when input condition is FALSE and sets the bit to 1 when input condition is FALSE

b. Output Latch
   sets a data table bit to 1 when input condition is TRUE; does nothing when input condition is FALSE

c. Output Unlatch
   sets a data table bit to 0 when input condition is TRUE; does nothing when input condition is FALSE
2. Timing and Counting Instructions

Timers

a. Timer On-Delay —(TON)— sets data table bit to 1 when timer times out; timer times while input condition is TRUE; timer is reset and data table bit is set to 0 when input condition is FALSE

b. Timer Off-Delay —(TOF)— sets data table bit to 1 when timer times out; timer times while input condition is FALSE; timer is reset and data table bit is 0 when input condition is TRUE

c. Retentive Timer —(RTO)—
   Reset
sets data table bit to 1 when timer times out; timer times while input condition is TRUE and retains accumulated time when input condition is FALSE

d. Retentive Timer —(RTR)—
   Reset
sets data table bit (that was set by a retentive timer) to 0 when input condition is TRUE

Timer increments: 1.0 second, 0.1 second, 0.01 second
Timer range: 0-999

Counters

a. Up Counter —(CTU)—
sets data table bit to 1 when counter counts up to preset value; increments accumulated count when input condition changes from FALSE to TRUE

b. Down Counter —(CTD)—
sets data table bit to 1 when counter counts down to preset value; decrements accumulated count when input condition changes from FALSE to TRUE

c. Counter Reset —(CTR)—
resets up counter count to 0 when input condition is TRUE

Counter range: 0-999
3. Arithmetic Instructions

a. Add — (+) — programmed with two Get instructions to add the values stored in two data table words and place result in a third word

b. Subtract — (-) — programmed with two Get instructions to subtract the values stored in two data table words and place result in a third word

a. Multiply — (X)(X) — programmed with two Get instructions to multiply the values stored in two data table words and place result in a third and fourth word

a. Divide — (:)(=) — programmed with two Get instructions to divide the values stored in two data table words and place result in a third and fourth word

4. Data Manipulation Instructions

a. Les — (<) — programmed with Get and Equ to compare two data table words; sets data table bit when comparison is TRUE

b. Equ — (=) — programmed with Get and Les to compare two data table words; sets data table bit when comparison is TRUE

c. Get Byte — (B) — programmed with Limit Test to compare a data table byte with upper and lower limit bytes obtained from a data table word

c. Limit Test — (L) — programmed with Get Byte to compare a data table byte with upper and lower limit bytes obtained from a data table word; sets data table bit when byte is equal to or between limit bytes
5. Data Transfer Instructions

a. Get —{G— programmed with a Put instruction to transfer the contents of one word in a data table to another memory word

b. Put —{(P)— programmed with a Get instruction to transfer the contents of one word in a data table to another memory word

6. Program Control Instructions

a. Master Control —(MCR)— two MCR instructions mark the beginning and end of a control zone in which all nonretentive outputs may be overridden; outputs are turned off when first MCR instruction is FALSE; outputs are under normal control when the MCR instruction is TRUE

b. Zone Control —(ZCL)— two ZCL instructions mark the beginning and end of a control zone in which all outputs may be overridden; outputs are held in their last state when first ZCL instruction is FALSE; outputs are under normal control when ZCL instruction is TRUE

c. Immediate Input —{I— updates one word (one I/O module) of input image table in advance of the normal scan sequence

d. Immediate Output —(IOT)— updates one I/O module from one word of output image table in advance of the normal scan sequence
ALLEN—BRADLEY PLC—2/30 INSTRUCTION SET

1. Relay Logic Instructions
   Contacts
   same as PLC—2/20
   Coils
   same as PLC—2/20

2. Timing and Counting Instructions
   Timers
   same as PLC—2/20
   Counters
   same as PLC—2/20 plus:
   a. Scan Counter —(SCT)— increments once per program scan while input condition is TRUE

3. Arithmetic Instructions
   same as PLC—2/20
4. Data Manipulation Instructions

same as PLC-2/20 plus:

a. Set Shift Bit
   SET SHIFT BIT sets a specified bit to 1

b. Reset Shift Bit
   RESET SHIFT BIT sets a specified bit to 0

c. Examine Off Shift Bit
   EXAMINE OFF SHIFT BIT examines a specified bit for an OFF condition

d. Examine On Shift Bit
   EXAMINE ON SHIFT BIT examines a specified bit for an ON condition

e. Bit Shift Left
   BIT SHIFT LEFT copies a bit from a data table word, shifts up to 999 consecutive bits left and copies the last bit to another data table word

f. Bit Shift Right
   BIT SHIFT RIGHT copies a bit from a data table word, shifts up to 999 consecutive bits right and copies the last bit to another data table word

g. Sequencer Input
   SEQUENCER INPUT compares up to four words of data with specified data table words

h. Sequencer Output
   SEQUENCER OUTPUT transfers up to four words of data to specified data table words

i. Sequencer Load
   SEQUENCER LOAD loads up to four words of data into a specified sequencer
5. Data Transfer Instructions

same as PLC-2/20 plus:

a. File to File Move FILE TO FILE MOVE copies a file of data table words to another file

b. Word to File Move WORD TO FILE MOVE copies a data table word to a file

c. File to Word Move FILE TO WORD MOVE copies one word from one file to another file

d. Shift File Up SHIFT FILE UP copies one data table word into the first word of a file, shifts all words in the file up one word position and copies the last file word to an output address

e. Shift File Down SHIFT FILE DOWN copies one data table word into the last word of a file, shifts all words in the file down one word position and copies the first file word to an output address

f. Block Transfer Read BLOCK TRANSFER READ programmed with BLOCK TRANSFER WRITE to transfer up to 64 words of data in one scan

g. Block Transfer Write BLOCK TRANSFER WRITE

6. Program Control Instructions

same as PLC-2/20
1. Relay Logic Instructions

Contacts
same as PLC-2/20

Coils
same as PLC-2/20

2. Timing and Counting Instructions

Timers
same as PLC-2/20

Counters
same as PLC-2/20

3. Arithmetic Instructions

same as PLC-2/20 plus:

These instructions are programmed with up to four Get instructions and process operands that may each be up to four words long.

a. Add EAF 01
b. Subtract EAF 02
c. Multiply EAF 03
d. Divide EAF 04
e. Square Root EAF 05
4. Data Manipulation Instructions

same as PLC-2/20 plus:

a. Sequencer Sequencer compares up to four words of data with specified data table words
   Input Input

b. Sequencer Sequencer transfers up to four words of data to specified data table words
   Output Output

c. Sequencer Sequencer loads up to four words of data into a specified sequencer
   Load Load

d. BCD to Binary Conversion
   EAF 13

e. Binary to BCD Conversion
   EAF 14

5. Data Transfer Instructions

same as PLC-2/20 plus:

a. File to File File 10 copies a file of data table words to another file
   Move Move

b. Word to File File 11 copies a data table word to a file
   Move Move

c. File to Word File 12 copies one word from one file to another file
   Move Move

d. Block Transfer Block programmed with BLOCK TRANSFER WRITE to transfer up to 64 words of data in one scan
   Transfer Read
   Transfer WRITE

e. Block Transfer
   Block
   Transfer WRITE

6. Program Control Instructions

same as PLC-2/20
1. Relay Logic Instructions

Contacts

a. Normally Open Contact
   - contact is open (open circuit) when data table bit is 0 and closed (short circuit) when data table bit is 1.

b. Normally Closed Contact
   - contact is closed (short circuit) when data table bit is 0 and open (open circuit) when data table bit is 1.

Coils

a. Coil
   - sets data table bit to 0 when input condition is FALSE and sets the bit to 1 when input condition is TRUE

b. Latch
   - LATCH programmed with a coil to cause it to latch when the input condition is TRUE; coil will remain on when input condition changes to FALSE; a second input rung in the network controls the unlatching of the coil

c. One Shot
   - (OS) sets a data table bit to 1 when input condition changes from FALSE to TRUE; sets the data table bit to 0 on next scan
2. Timing and Counting Instructions

Timers

a. Timer

Timer TIMER sets data table bit to 1 when timer times out; timer times while input condition on input rung is TRUE; it retains accumulated time while input condition on input rung is FALSE; timer resets and data table bit is set to zero when input condition on reset rung is TRUE.

Timer increments: 1.0 second, 0.1 second, 0.01 second
Timer range: 0-999

Counters

a. Up Counter

Up Counter COUNTER programmed with a One Shot; sets data table bit to 1 when counter counts up to preset value; increments accumulated count when input condition on input rung changes from FALSE to TRUE; counter resets to zero and data table bit is set to zero when input condition on reset rung is TRUE.

b. Down Counter

Down Counter COUNTER programmed with a One Shot; sets data table bit to 1 when counter counts down to zero; increments accumulated count when input condition on input rung changes from FALSE to TRUE; counter resets to preset value and data table bit is set to zero when input condition on reset rung is TRUE.

Counter range: 0-999 (0-65,535 with register preset)
3. Arithmetic Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. Add A+B</td>
<td>adds two storage registers and places the result in a third; the operation is performed if the input condition is TRUE; outputs a TRUE condition if the sum exceeds 65,535</td>
</tr>
<tr>
<td>b. Subtract A-B</td>
<td>subtracts two storage registers and places the result in a third; the operation is performed if the input condition is TRUE; outputs a TRUE condition if the difference is zero or negative</td>
</tr>
<tr>
<td>c. Multiply MULT</td>
<td>multiplies the signed values in two storage registers and places the signed result in two storage registers; the operation is performed if the input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td>d. Divide DIV</td>
<td>divides the signed values in two storage registers and places the signed result in two storage registers; the operation is performed if the input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td>e. Single Precision Add ADDX</td>
<td>adds the signed values in two storage registers and places the signed result in a third; the operation is performed if the input condition is TRUE; outputs a TRUE condition if sum exceeds 32,767</td>
</tr>
<tr>
<td>f. Single Precision Subtract SUBTX</td>
<td>subtracts the signed values in two storage registers and places the result in a third; the operation is performed if the input condition is TRUE; outputs a TRUE condition if difference exceeds -32,768</td>
</tr>
<tr>
<td>g. Double Precision Add DPADD</td>
<td>adds two signed values each stored in two storage registers and places the signed result in two storage registers; the operation is performed if the input condition is TRUE; outputs a TRUE condition if sum exceeds 2,147,483,647</td>
</tr>
</tbody>
</table>
h. Double
Precision Subtract

DPSUBT subtracts two signed values each stored in two storage registers and places the signed result in two storage registers; the operation is performed if the input condition is TRUE; outputs a TRUE condition if result exceeds 
\[-2,147,483,647\]
4. Data Manipulation Instructions

a. Compare COMPARE compares two storage register values; the operation is performed if the input condition is TRUE; outputs a TRUE condition if the two values are equal.

b. Greater Than GR THAN compares two signed values each stored in two storage registers; performs operation if the input condition is TRUE; outputs a TRUE condition if compare is TRUE.

c. BCD to Binary BCD TO BIN converts an I/O register to binary and places result in a storage register; performs operation when input condition is TRUE; outputs a TRUE condition if BCD pattern represents a digit above 9.

d. Binary to BCD BIN TO BCD converts a value in a storage register to BCD and places result in a specified I/O register; performs operation when input condition is TRUE; outputs a TRUE condition if binary value exceeds 9,999.

e. AND AND performs a logical AND operation on the bits in two sets of storage locations of a specified number of words and places the result in a third set of storage locations.

f. IOR IOR performs a logical inclusive OR operation on the bits in two sets of storage locations of a specified number of words and places the result in a third set of storage locations.

g. EOR EOR performs a logical exclusive OR operation on the bits in two sets of storage locations of a specified number of words and places the result in a third set of storage locations.

h. Invert INVERT transfers the contents of one set of storage locations to another set of storage locations and performs a logical NOT operation on the contents in the second set.
i. Compare COMPARE compares the bits of two sets of storage locations using the bits in a third set of storage locations as a mask

j. Bit Set SET sets the specified bit in a set of storage locations to 1

k. Bit Clear CLEAR sets the specified bit in a set of storage locations to 0

l. Shift Right SHIFT R shifts right a specified number of bits beginning at a specified storage location; zeros are shifted into the open bit locations

m. Shift Left SHIFT L shifts left a specified number of bits beginning at a specified storage location; zeros are shifted into the open bit locations
### 5. Data Transfer Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>I/O Register to Storage Register</strong></td>
<td><strong>I/O TO R</strong> transfers the contents of an I/O register to a storage register when the input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td><strong>Storage Register to I/O</strong></td>
<td><strong>R TO I/O</strong> transfers the contents of a storage register to an I/O register when the input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td><strong>Shift Register</strong></td>
<td><strong>SHIFT</strong> shifts the bits in an I/O register one position toward the higher register number when input condition is TRUE; sets open bit position to 0</td>
</tr>
<tr>
<td><strong>Source to Table Move</strong></td>
<td><strong>S TO T</strong> loads a table of up to 255 memory words with a value copied from a single storage location when input condition is TRUE; one transfer is performed each scan; outputs a TRUE condition when data is loaded into last word of table</td>
</tr>
<tr>
<td><strong>Table to Destination Move</strong></td>
<td><strong>T TO D</strong> transfers one word from a table to a single storage location when input condition is TRUE; one transfer is performed each scan; outputs a TRUE condition when last word of table is transferred</td>
</tr>
<tr>
<td><strong>Table to Table Move</strong></td>
<td><strong>T TO T</strong> transfers the contents of one table to another table when input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td><strong>A to B Move</strong></td>
<td><strong>A TO B</strong> transfers one word of data from one storage location to another when input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td><strong>Move Left Half</strong></td>
<td><strong>L8</strong> transfers the high order byte of a word from one storage location to another when input condition is TRUE; always outputs a TRUE condition</td>
</tr>
<tr>
<td><strong>Move Right Half</strong></td>
<td><strong>R8</strong> transfers the low order byte of a word from one storage location to another when input condition is TRUE; always outputs a TRUE condition</td>
</tr>
</tbody>
</table>
j. Block Move

 BLOCK
 MOVE
 transfers seven words of data from program memory to seven consecutive storage locations

k. Add to Top

 ADD TO
 TOP
 transfers one word of data to the first position of a table of specified length and shifts all other table values when input condition is TRUE; always outputs a TRUE condition

l. Remove from Bottom

 REMOVE
 FROM
 BOTTOM
 transfers one word of data from the last position of a table of specified length to a storage location and shifts all other table values when input condition is TRUE; always outputs a TRUE condition

m. Remove from Top

 REMOVE
 FROM
 TOP
 operates similarly to Remove from Bottom but removes data word from top of table

n. Ascending Sort

 SORT
 sorts a table of specified length and arranges words in ascending order of value
6. Program Control Instructions

a. Do I/O  DO I/O  causes controller to update a series of I/O registers

b. Suspend I/O  SUSP I/O  inhibits the normal updating of I/O registers

c. Master Control Relay  MCR  specifies a number of networks to be bypassed during a program scan; all bypassed outputs are turned off when the input condition to the MCR is TRUE; outputs are under normal control when the input condition is FALSE

d. Skip  SKIP  specifies a number of networks to be bypassed during a program scan; all bypassed outputs are held in their last state when the input condition to the SKIP is TRUE; outputs are under normal control when the input condition is FALSE

e. End Sweep  ENDSW  marks the end of the main program

f. Do Subroutine  DO SUB  causes program scan to jump to a subroutine

g. Return  RETURN  causes program scan to return to main program
GoLD lOClIC inStruCTioNS

1. Relay Logic Instructions

Contacts

a. Normally Open Contact  
   contact is open (open circuit) when data table bit is 0 and closed (short circuit) when data table bit is 1.

b. Normally Closed Contact  
   contact is closed (passes power) when data table bit is 0 and open (open circuit) when data table bit is 1.

c. Positive Transitional Contact  
   contact closes (passes power) for one scan when its data table bit changes from 0 to 1

d. Negative Transitional Contact  
   contact closes (passes power) for one scan when its data table bit changes from 0 to 1

Coils

a. Coil  
   sets data table bit to 0 when input condition is FALSE and sets the bit to 1 when input condition is TRUE

b. Latch  
   sets latch bit to 1 when latch rung is TRUE; sets latch bit to 0 when reset rung is TRUE; top output rung passes power if set rung is TRUE only; bottom output rung passes power if reset rung is TRUE (set rung can be TRUE or FALSE)
2. Timing and Counting Instructions

Timer

a. Timer \(\text{Tx.x}\) timer times when input rung and enable rung are TRUE; timer resets to zero when enable rung is FALSE; top output rung passes power when accumulated time reaches preset value; bottom output rung passes power when accumulated time does not equal preset value;

Timer increments: 1.0 second, 0.1 second, 0.01 second
Timer range: 0-9,999

Counters

a. Up Counter \(\text{UCTR}\) increments accumulated count once each scan when input rung and enable rung are TRUE; count value resets to zero when enable rung is FALSE; output rung passes power when accumulated count equals preset value

b. Down Counter \(\text{DCTR}\) decrements accumulated count once each scan when input rung and enable rung are TRUE; count value resets to zero when enable rung is FALSE; output rung passes power when accumulated count equals zero

Counter range: 0-9,999
3. Arithmetic Instructions

a. Add

ADD adds two values (constant, sequencer register, holding register) and places result in sequencer or holding register when input rung is TRUE; adds 1 to result when carry rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when result exceeds 9,999; bottom output rung passes power when an operand exceeds 9,999

b. Double Precision Add

DADD adds two signed values (constant, double precision register) and places result in double precision register when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when underflow or overflow occurs

c. Subtract

SUB subtracts two values (constant, sequencer register, holding register) and places result in sequencer or holding register when input rung is TRUE; subtracts 1 from result when borrow rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when result is less than zero; bottom output rung passes power when an operand exceeds 9,999

d. Double Precision Subtraction

DSUB subtracts two signed values (constant, double precision register) and places result in double precision register when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when underflow or overflow occurs

e. Multiply

MULT multiplies two values (constant, sequencer register, holding register) and places result in two consecutive holding registers when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when an operand exceeds 999
f. Double Precision Multiply
   DMUL multiplies two signed values (constant, double precision register) and places result in a double precision register when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when an underflow or overflow occurs.

g. Divide
   DIV divides two values (constant, sequencer register, holding register) and places quotient in a holding register and remainder in next holding register when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when an operand exceeds 999 or divisor equals zero.

h. Double Precision Divide
   DDIV divides two signed values (constant, double precision register) and places result in a double precision register when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when an underflow or overflow occurs or divisor equals zero.
4. Data Manipulation Instructions

a. Bit Shift Left  

BSHL  
shifts bits in a matrix left one bit per scan when input rung is TRUE; sets open bit position to 1 if carry in rung is TRUE; sets open bit position to 0 if carry in rung is FALSE; top output rung passes power if input rung is TRUE; bottom output rung passes power if bit shifted out of matrix is 1

b. Bit Shift Right  

BSHR  
shifts bits in a matrix right one bit per scan when input rung is TRUE; sets open bit position to 1 if carry in rung is TRUE; sets open bit position to 0 if carry in rung is FALSE; top output rung passes power if input rung is TRUE; bottom output rung passes power if bit shifted out of matrix is 1

c. Bit Sense  

SENS  
examines and reports state of bits in a matrix; increments pointer and examines one bit per scan when input rung is TRUE; top output rung passes power if input rung is TRUE; middle output rung passes power if examined bit is 1; bottom output rung passes power if pointer is past end of matrix or equals zero

d. Bit Modify  

MBIT  
alters states of bits in a matrix; sets one bit per scan to one and increments pointer if input rung and set rung are TRUE; sets one bit per scan to zero and increments pointer if input rung is TRUE and set rung is FALSE; top output rung passes power if input rung is TRUE; middle output rung passes power if the bit modified is a disabled discrete; bottom output rung passes power if pointer is past matrix end or equals zero
e. Test  
TEST compares two unsigned values (constant, sequencer register, holding register, double precision register) when input rung is TRUE; top output rung passes power when first value is greater than second; middle output rung passes power when first value equals second; bottom output rung passes power when first value is less than second.

f. Sequencer  
SEQ allows sequential type control of a single point; increments a sequencer register when input rung and enable rung are TRUE; top output rung passes power when value in sequencer register equals preset; bottom output rung passes power when input rung is TRUE and enable rung is FALSE.

g. Drum  
DRUM allows sequential type control of many points; one register of source table is moved into destination table per scan when input rung is TRUE; top output rung passes power when input rung is TRUE; bottom output rung passes power when input rung is TRUE and pointer is past end of source table.

h. Single to Double Precision Conversion  
X TO 5 converts two single precision unsigned values (constant,input register, holding register) into a double precision value in a double precision register when input rung is TRUE; makes result negative if set negative rung is TRUE; top output rung passes power if input rung is TRUE; bottom output rung passes power if either operand exceeds 9,999.

i. Double to Single Precision Conversion  
5 TO X converts a double precision value (double precision register) to two single precision values in two holding registers when input rung is TRUE; top output rung passes power if double precision value is negative; bottom output rung passes power if double precision value exceeds 99,999,999.
5. Data Transfer Instructions

a. Table to Register Move  
   T TO R transfers one data word from a table of registers to a holding register and increments pointer to next data word each scan when input rung is TRUE; pointer resets to zero and does not increment when reset rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table; bottom output rung passes power when pointer is at or past end of table.

b. Register to Table Move  
   R TO T transfers the contents of a register to one register in a table of registers and increments pointer to next table register each scan when input rung is TRUE; pointer resets to zero and does not increment when reset rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table; bottom output rung passes power when pointer is at or past end of table.

c. Block Move  
   BLKM copies an entire table of registers to another table on one scan when input rung is TRUE; output rung passes power when input rung is TRUE.

6. Program Control Instructions

a. Skip  
   SKP specifies a number of networks to be bypassed during a program scan; all bypassed outputs are held in their last state when input rung is TRUE; outputs are under normal control when input rung is FALSE.
1. Relay Logic Instructions

Contacts
same as MODICON 884

Coils
same as MODICON 884

2. Timing and Counting Instructions

Timer
same as MODICON 884

Counters
same as MODICON 884 except MODICON 584 counters have a bottom output rung which operates as follows:

Up Counter - passes power if accumulated count is less than preset

Down Counter - passes power if accumulated count is greater than zero and less than or equal to preset
3. Arithmetic Instructions

a. Add

Add (ADD) adds two values (constant, input register, holding register) and places result in holding register when input rung is TRUE; output rung passes power when result exceeds 9,999;

b. Subtract

Subtract (SUB) subtracts two values (constant, input register, holding register) and places result in holding register when input rung is TRUE; top output rung passes power when result is positive; middle output rung passes power when result is zero; bottom output rung passes power when result is negative.

c. Multiply

Multiply (MULT) multiplies two values (constant, input register, holding register) and places result in two consecutive holding registers when input rung is TRUE; top output rung passes power when input rung is TRUE.

d. Divide

Divide (DIV) divides two values (constant, input register, holding register) and places quotient in a holding register and remainder in next holding register when input rung is TRUE; remainder value is a decimal fraction if middle input rung is TRUE, an integer if FALSE; top output rung passes power if division is successful; middle output passes power if quotient exceeds 9,999; bottom output rung passes power if divisor equals 0.
4. Data Manipulation Instructions

a. Logical AND

AND performs AND operation on two matrices and places result in the second when input rung is TRUE; output rung passes power if input rung is TRUE.

b. Logical OR

OR performs OR operation on two matrices and places result in the second when input rung is TRUE; output rung passes power if input rung is TRUE.

c. Logical XOR

XOR performs XOR operation on two matrices and places result in the second when input rung is TRUE; output rung passes power if input rung is TRUE.

d. Logical Complement

COMP performs NOT operation on a matrix and places result in another matrix when input rung is TRUE; output rung passes power if input rung is TRUE.

d. Logical Compare

CMPR compares two matrices bit by bit, until two bits do not agree, when input rung is TRUE; places pointer location of miscompare bits in a holding register; resets pointer value to zero when reset rung is TRUE; top output rung passes power if input rung is TRUE; middle output rung passes power if miscompare is found; bottom output rung passes power if miscompare bit in first matrix is a 1.

e. Bit Sense

SENS examines and reports state of bits in a matrix; examines one bit per scan when input rung is TRUE; increments pointer when middle input rung is TRUE; resets pointer when bottom input rung is TRUE; top output rung passes power if input rung is TRUE; middle output rung passes power if examined bit is 1; bottom output rung passes power if pointer is past end of matrix.
f. Bit Modify

MBIT alters states of bits in a matrix; sets one bit per scan when top input rung is TRUE; sets bit to one if middle input rung is TRUE; sets bit to zero if middle input rung is FALSE; increments pointer if bottom input rung is TRUE; top output rung passes power if input rung is TRUE; middle output rung passes power if middle input is TRUE; bottom output rung passes power if pointer is past matrix end.

g. Bit Rotate

BROT shifts or rotates bits in a matrix, one bit per scan, when input rung is TRUE; middle input rung controls direction of shift; rotate operation is performed if bottom input rung is TRUE; top output rung passes power if input rung is TRUE; middle output rung passes power if bit shifted out of matrix is 1.
5. Data Transfer Instructions

a. Table to Register Move

T TO R

transfers one data word from a table of registers to a holding register and increments pointer to next data word each scan when input rung is TRUE; pointer does not increment when middle input rung is TRUE; pointer resets to zero when reset rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table

b. Register to Table Move

R TO T

transfers the contents of a register to one register in a table of registers and increments pointer to next table register each scan when input rung is TRUE; pointer does not increment when middle input rung is TRUE; pointer resets to zero when reset rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table

c. Table to Table Move

T TO T

copies an entire table of registers or discretes to a table of registers one word per scan when input rung is TRUE; pointer does not increment when middle input rung is TRUE; pointer resets to zero when reset rung is TRUE; output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table

d. Block Move

BLKM

copies an entire table of registers to another table on one scan when input rung is TRUE; top output rung passes power when input rung is TRUE

e. First In

FIN

copies data from a source register into the first position of a table and increments pointer when input rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table; bottom output rung passes power when pointer is zero
f. First Out FOUT removes oldest data from a table, places it in a holding register and decrements pointer when input rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when pointer is at end of table; bottom output rung passes power when pointer is zero.

g. Table Search SRCH searches a table of registers for a specified value; examines one register and increments pointer each scan when input rung is TRUE; search begins at first register if middle input rung is FALSE, at pointer register if middle input rung is TRUE; top output rung passes power when input rung is TRUE; middle output rung passes power when a match is found.

6. Program Control Instructions

same as MODICON 884
The vita has been removed from the scanned document
TOWARD THE DEVELOPMENT OF A UNIVERSAL PROGRAMMING/DOCUMENTATION SYSTEM FOR PROGRAMMABLE CONTROLLERS ON A HOST MICROCOMPUTER

by

John Andrew Snader

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Science in Industrial Engineering and Operations Research

October 4, 1985

Blacksburg, Virginia
APPENDIX F. SYSTEM PROGRAM SOURCE LISTING

C BLOCK DATA QATRIB

C This common block allocates variables that are used to represent
C different attribute combinations for the IBM Color Graphics Adapter.

C PAGE - current active video page

    integer*2 PAGE, ATRIB0, ATRIB1, ATRIB2, ATRIB3, ATRIB4,
    * ATRIB5, ATRIB6, ATRIB7, ATRIB8, ATRIB9, ATRIBA
    common /ATRIBS/ PAGE, ATRIB0, ATRIB1, ATRIB2, ATRIB3, ATRIB4,
    * ATRIB5, ATRIB6, ATRIB7, ATRIB8, ATRIB9, ATRIBA
    data PAGE /0/
    end

C BLOCK DATA GLADDR

C CONSTANTS:
C RON - cursor positioning constant for screen row
C COL - cursor positioning constant for screen column

C VARIABLES:
C ELEMAX - maximum number of elements in the element data linked list
C ELETOT - current number of elements in the element data linked list
C ELMFREE - pointer to first free record in the element data linked list
C ELHEAD - pointer to first entry in the element data linked list
C ELMFULL - boolean flag to indicate that element data linked list is full
C ELEPTR(1) - pointer to the table position of the element at the current
C and points to table position of the preceding element in the network
C ELMNTx - element data table:
C ELMNTC(1,1) - element type
C ELMNTC(1,2) - row coordinate of element
C ELMNTC(1,3) - column coordinate of element
C ELMNTC(1,4) - link data
C ELMNTI(1) - pointer to the preceding element in the table
C ELMNTI(2) - pointer to the following element in the table
C ELMNTI(3) - element reference number or pointer to box data record
C ENDLDR - flag to indicate that displayed network is last network in ladder
C MNFREE - pointer to first free record in the network data linked list
C MNHEAD - pointer to first entry in the network data linked list
C MNTAIL - pointer to last entry in the network data linked list
C MNFULL - boolean flag to indicate that network data linked list is full
C NTHMAX - maximum number of networks in the network data linked list
C NTHNUM - number of the current network being displayed
C NTNTOT - current total number of networks in the ladder
C NTNPTR - index number of the current ladder network
C NETRK - network data table:
C NETRK(1) - pointer to the first element of the ladder network
C NETRK(2) - pointer to the last element of the ladder network
C NETRK(3) - number of rungs in the ladder network
C NETRK(4) - pointer to the preceding ladder network in the table
C NETRK(5) - pointer to the following ladder network in the table
C ABRSRNG - absolute rung number of first rung in current network
C RUNG - ladder rung position of cursor
C COLUMN - ladder column position of cursor
C ROHCNT - number of elements in each row of the current network
C KEY - identifier of current prompt
C SBXMAX - maximum number of small boxes
C SBXTOT - current number of small boxes
C LBXMAX - maximum number of large boxes
C LBXTOT - current number of large boxes

character*1 ELMNTC(4096,4)
integer*2 ABSRNG, COL(12), COLUMN, ELEMAX, ELEPTR(7,12), ELETOT,
  * ELFREE, ELHEAD, ELMNTI(4096,3), KEY, LBXMAX, LBXTOT,
  * NETHRK(2048,5), NTMNX, NTNQJM, NTHPTR, NTNTOT, NNFREE,
  * NHHEAD, NNTAIL, ROW(7), RWCNT(7), RUNG, SBXMAX, SBXTOT
logical*2 ELFULL, ENDLDR, NMFULL

common /LADDER/ ABSRNG, COL, COLUMN, ELEMAX, ELEPTR, ELETOT,
  * ELFREE, ELFULL, ELHEAD, ELMNTC, ELMNTI, ENDLDR,
  * KEY, LBXMAX, LBXTOT, NETHRK, NTMNX, NTNQJM,
  * NTHPTR, NTNTOT, NNFREE, NMFULL, NHHEAD, NNTAIL,
  * ROW, ROWCNT, RUNG, SBXMAX, SBXTOT

data COL /6,12,18,24,30,36,42,48,54,60,66,74/
data ROW /2,5,8,11,14,17,20/
data ELEMAX, LBXMAX, NTMNX, SBXMAX /4096,30,2048,128/
data ELETOT, LBXTOT, NTNTOT, SBXTOT /4*0/
end

C BLOCK DATA QBOXES

C SBOX(*,1) - reference number of the small box (1..128)
C SBOX(*,2) - preset to which timer times or counter counts
C SBOX(*,3) - reference number of coil to be energized by U/D counter
C when it reaches a zero count
C SBOXC(*) - zero coil type (Y or C)
C LBOX(*,1) - reference number of the large box (1..30)
C LBOX(*,2) - step at which the drum begins its sequence after being reset
C LBOX(*,3) - time interval between counts (seconds/count)
C LBOX(*,4) through
C LBOX(*,19) - number of counts duration for each of 16 steps (counts/step)
C LBOX(*,20) through
C LBOX(*,34) - reference numbers of the 15 controlled output coils
C LBOX(*,35) through
C LBOX(*,50) - drum masks for the 16 controlled output coil steps
C LBOXC(*,1) - output coil type (Y or C)
C through
C LBOXC(*,15)
C TYPREF - type (X,Y) cross reference for discrete elements
C SBXREF - pointer to element data list record given small box reference number
C LBXREF - pointer to element data list record given large box reference number

character*1 LBOXC(30,15), SBOXC(128), TYPREF(1023)
integer*2 LBOX(30,50), LBXREF(30), SBOX(128,3), SBXREF(128)

common /BOXES/ LBOX,LBOXC,LBXREF,SBOX,SBOXC,SBXREF,TYPREF
end

C PROGRAM MAIN

$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QLADDER.BLK'

integer*2 BLINK, BOTTOM, C, J, Q1, Q2, R, TOP
logical*2 NENDAT, NODATA
data BLINK,BOTTOM,C,R,TOP /128,18,36,8,8/
data NENDAT,NODATA /./false.,true./
call SETMOD(3)
call INITLL
call CRTTYP
call CLS(ATRIB1)
call SETPAG(3)

C MENU1  page 0
C MENU2  page 1
C EDITOR  page 2

C (set up EDITOR screen)
PAGE=2
call LOCATE(PAGE,25,1)
call MRCHAT(PAGE,' ',ATRIB5,80)
call PRINT(PAGE,25,2,'Network',ATRIB5,7)
call PRINT(PAGE,25,16,'Row',ATRIB5,3)
call PRINT(PAGE,25,23,'Column',ATRIB5,6)
KEY=2
call PROMPT(KEY)

C (set up MENU2 screen)
PAGE=1
call LOCATE(PAGE,1,4)
call MRCHAT(PAGE,char(201),ATRIB1,1)
call MRCHAT(PAGE,char(205),ATRIB1,72)
call LOCATE(PAGE,1,77)
call MRCHAT(PAGE,char(187),ATRIB1,1)
do 10 J=2,24
   call LOCATE(PAGE,J,4)
call MRCHAT(PAGE,char(186),ATRIB1,1)
call LOCATE(PAGE,J,77)
call MRCHAT(PAGE,char(186),ATRIB1,1)
10 continue
call LOCATE(PAGE,25,4)
call MRCHAT(PAGE,char(200),ATRIB1,1)
call MRCHAT(PAGE,char(205),ATRIB1,72)
call LOCATE(PAGE,25,77)
call MRCHAT(PAGE,char(188),ATRIB1,1)
call LOCATE(PAGE,3,4)
call MRCHAT(PAGE,char(199),ATRIB1,1)
call MRCHAT(PAGE,char(196),ATRIB1,72)
call LOCATE(PAGE,3,77)
call MRCHAT(PAGE,char(182),ATRIB1,1)
call LOCATE(PAGE,6,26)
call MRCHAT(PAGE,char(201),ATRIB1,1)
call MRCHAT(PAGE,char(205),ATRIB1,28)
call LOCATE(PAGE,6,55)
call MRCHAT(PAGE,char(187),ATRIB1,1)
call LOCATE(PAGE,7,26)
call MRCHAT(PAGE,char(186),ATRIB1,1)
call PRINT(PAGE,7,27,' LADDER FILE ',ATRIB4,13)
call PRINT(PAGE,7,40,' | BOOLEAN FILE ',ATRIB1,15)
call MRCHAT(PAGE,char(186),ATRIB1,1)
call LOCATE(PAGE,8,26)
call MRCHAT(PAGE,char(200),ATRIB1,1)
call MRCHAT(PAGE,char(205),ATRIB1,28)
call LOCATE(PAGE,8,55)
call MRCHAT(PAGE,char(188),ATRIB1,1)
call LOCATE(PAGE,6,40)
call MRCHAT(PAGE,char(209),ATRIB1,1)
call LOCATE(PAGE,8,40)
call MRCHAT(PAGE,char(207),ATRIB1,1)
call PRINT(PAGE,10,26,'Use TAB Key To Select File Type',ATRIB1,31)
call PRINT(PAGE,15,23,'Enter name of file to ',ATRIB1,22)
call PRINT(PAGE,24,26,' Press F1 To Return To Menu ',ATRIB1,30)

C (set up MENU1 screen)
PAGE=0
call LOCATE(PAGE,1,4)
call WRCHEAR(PAGE,char(201),1)
call WRCHEAR(PAGE,char(205),72)
call LOCATE(PAGE,1,77)
call WRCHEAR(PAGE,char(187),1)
do 20 J=2,24
   call LOCATE(PAGE,J,4)
call WRCHEAR(PAGE,char(186),1)
call LOCATE(PAGE,J,77)
call WRCHEAR(PAGE,char(186),1)
20 continue
call LOCATE(PAGE,25,4)
call WRCHEAR(PAGE,char(200),1)
call WRCHEAR(PAGE,char(205),72)
call LOCATE(PAGE,25,77)
call WRCHEAR(PAGE,char(188),1)
call LOCATE(PAGE,3,4)
call WRCHEAR(PAGE,char(199),1)
call WRCHEAR(PAGE,char(196),72)
call LOCATE(PAGE,3,77)
call WRCHEAR(PAGE,char(182),1)
call PRINT(PAGE,2,21,'PROGRAMMABLE CONTROLLER PROGRAMING SYSTEM',*
   ATRIB3,41)
call PRINT(PAGE,4,17,'PRESS ',ATRIB1,6)
call WRCHEAR(PAGE,char(24),ATRIB3,1)
call PRINT(PAGE,4,24,' TO MOVE ARROW UP OR ',ATRIB1,21)
call WRCHEAR(PAGE,char(25),ATRIB3,1)
call PRINT(PAGE,4,46,' TO MOVE ARROW DOWN',ATRIB1,19)
call PRINT(PAGE,5,26,'PRESS ',ATRIB1,6)
call WRCHEAR(PAGE,char(17),ATRIB3,1)
call WRCHEAR(PAGE,char(196),ATRIB3,1)
call WRCHEAR(PAGE,char(197),ATRIB3,1)
call PRINT(PAGE,5,35,' TO SELECT AN OPTION',ATRIB1,20)
call PRINT(PAGE,TOP,C+2,'EDITOR',ATRIB1,6)
call PRINT(PAGE,TOP+2,C+2,'LOAD',ATRIB3,4)
call PRINT(PAGE,TOP+4,C+2,'SAVE',ATRIB1,4)
call PRINT(PAGE,TOP+6,C+2,'UPLOAD',ATRIB1,6)
call PRINT(PAGE,STOP,B+2,'DOWNLOAD',ATRIB1,8)
call PRINT(PAGE,STOP+10,C+2,'QUIT',ATRIB1,4)
do 30 J=1,3
   call LOCATE(PAGE,19+J,18)
call WRCHEAR(PAGE,' ',ATRIB9,46)
30 continue
call HELP1(1)
call SETPAG(0)
100 call LOCATE(PAGE,20,18)
call WRCHEAR(PAGE,' ',46)
   if (R .eq. TOP) then
      call HELP1(1)
   elseif (R .eq. TOP+2) then
      call HELP1(2)
   elseif (R .eq. TOP+4) then
      call HELP1(3)
   elseif (R .eq. TOP+6) then
      call HELP1(4)
   elseif (R .eq. TOP+8) then
      call HELP1(5)
   else
      call HELP1(6)
   endif
200 call LOCATE(PAGE,R,C)
call WRCHEAR(PAGE,char(16),ATRIB8+BLINK,1)
call INKEY(Q1,Q2)

Appendix F. SYSTEM PROGRAM SOURCE LISTING 155
call LOCATE(PAGE,R,C)
call WRCHAT(PAGE,' ',ATRIB8,1)
if (Q1 .eq. 0) then
  C {EXTENDED CODE KEYS}
  if (Q2 .eq. 72) then
    C {cursor up}
    if (R .eq. TOP) then
      R=BOT
    else
      R=R-2
    endif
  elseif (Q2 .eq. 80) then
    C {cursor down}
    if (R .eq. BOTTOM) then
      R=TOP
    else
      R=R+2
    endif
  else
    call BEEP(1,1)
call ERRMSG(24,1)
goto 200
endif
else
  C {ASCII KEYS}
  else
    if (Q1 .eq. 13) then
      C {carriage return}
      if (R .eq. TOP) then
        call EDITOR(NODATA,NEHDAT)
      elseif (R .eq. TOP+2) then
        call MENU2(.false.,NODATA,NEHDAT)
        if (NEHDAT) call EDITOR(NODATA,NEHDAT)
      elseif (R .eq. TOP+4) then
        if (NODATA) then
          call BEEP(1,1)
call ERRMSG(24,6)
        else
          call MENU2(.true.,NODATA,NEHDAT)
        endif
      elseif (R .eq. TOP+6) then
        call UPLOAD(NODATA)
call BEEP(1,1)
call ERRMSG(24,2)
goto 200
      elseif (R .eq. TOP+8) then
        call DNLOAD
        call BEEP(1,1)
call ERRMSG(24,2)
goto 200
      else
        call CLS(ATRIB1)
call CSR
      end
    endif
else ((Q1 .eq. 69) .or. (Q1 .eq. 101)) then
  call EDITOR(NODATA,NEHDAT)
  R=TOP
  call HELP1(1)
else ((Q1 .eq. 76) .or. (Q1 .eq. 108)) then
  call MENU2(.false.,NODATA,NEHDAT)
  if (NEHDAT) call EDITOR(NODATA,NEHDAT)
  R=TOP+2
  call HELP1(2)
else ((Q1 .eq. 83) .or. (Q1 .eq. 115)) then
  R=TOP+4
  if (NODATA) then
    call HELP1(3)
call BEEP(1,1)
call ERRMSG(24,6)

Appendix F. SYSTEM PROGRAM SOURCE LISTING
else
  call MENU2(.true.,NODATA,NENDAT)
call HELP1(3)
endif
elseif ((Q1 .eq. 85) .or. (Q1 .eq. 117)) then
  R=TOP+6
call HELP1(4)
call UPLOAD(NODATA)
call BEEP(1,1)
call ERRMSG(24,2)
goto 200
elseif ((Q1 .eq. 68) .or. (Q1 .eq. 100)) then
  R=TOP+8
call HELP1(5)
call DNLOAD
call BEEP(1,1)
call ERRMSG(24,2)
goto 200
elseif ((Q1 .eq. 81) .or. (Q1 .eq. 113)) then
  call CLSIATRIB)
call CSRON
stop
else
  call BEEP(1,1)
call ERRMSG(24,1)
goto 200
endif
PAGE=0
call SETPAG(0)
endif
goto 100
dend

SUBROUTINE MENU2(TODISK,NODATA,NENDAT)
$INCLUDE:'QATRIB.BLK'
character*1 Q(8), EXT(4), TEMP(14)
character*2 DRIVE
character*14 FILSPC
equivalence (TEMP(1),DRIVE), (FILSPC,TEMP)
integer*2 FIELD, FLDCOL, FLDROM, H, J, K, Q1, Q2
logical*2 NENDAT, NODATA, OKFILE, TODISK
EXT(1)=.'.'
EXT(2)=.'L'
EXT(3)=.'D'
EXT(4)=.'R'
PAGE=1
call PRINT(PAGE,7,41,' BOOLEAN FILE ',ATRIB1,14)
call PRINT(PAGE,7,27,' LADDER FILE ',ATRIB4,13)
if (TODISK) then
call PRINT(PAGE,2,31,' SAVE FILE TO DISK ',ATRIB3,19)
call PRINT(PAGE,15,45,'save:',ATRIB1,5)
else
call PRINT(PAGE,2,31,'LOAD FILE FROM DISK',ATRIB3,19)
call PRINT(PAGE,15,45,'load:',ATRIB1,5)
endif
call SETPAG(1)
100 H=0
FIELD=8
FLDROM=15
FLDCOL=51
do J=1,8
   Q(J)= ' '
10 continue
 do 20 J=3,14
   TEMP(J)= ' '
20 continue
200 call LOCATE(PAGE,FLDROM,FLDCOL)
call HRCHAT(PAGE,char(177),ATRIB3,8)
300 if (H.lt. FIELD) then
   call LOCATE(PAGE,FLDROM,FLDCOL+H)
call CSRON
endif
call INKEY(Q1;Q2)
if (Q1 .eq. 0) then
   if (Q2 .eq. 59) then
      call CSROFF
      return
   else
      call BEEP(FLDROM,FLDCOL+H)
call ERRMSG(17,1)
   endif
else
   if (Q1 .eq. 9) then
      call CSROFF
      if (EXT(1) .eq. 0) then
         call BEEP(FLDROM,FLDCOL+H)
call ERRMSG(17,5)
call CSRON
goto 200
      else
         goto 400
      endif
   elseif (Q1 .eq. 13) then
      if (Q2 .eq. char(0)) then
         call BEEP(FLDROW,FLDCOL)
call ERRMSG(17,5)
call CSRON
goto 200
      elseif (Q1 .eq. 13) then
         if (Q2 .eq. 0) then
            call BEEP(FLDROM,FLDCOL+H-1)
call ERRMSG(17,5)
else if (Q1 .eq. 8) then
   (input field full)
call BEEP(FLDROW,FLDCOL+H-1)
call ERRMSG(17,5)
else if (Q1 .eq. 8) then
   (no previous input)
call BEEP(FLDROM,FLDCOL)
call ERRMSG(17,1)
else
   (previous input)
   Q2=char(0)
call LOCATE(PAGE,FLDROW,FLDCOL+H-1)
call HRCHAT(PAGE,' ',ATRIB3,1)
call LOCATE(PAGE,FLDROW,FLDCOL+H-1)
if (H .eq. FIELD) then
   (erase blinking carriage return)
call LOCATE(PAGE,FLDROM,FLDCOL+FIELD+1)
call MRCHAT(PAGE,' ',ATRIB1,3)
endif
H=H-1
endif
elseif ((Q1 .lt. 33) .or. (Q1 .gt. 126) .or. (Q1 .eq. 34) 
* .or. ((Q1 .gt. 41) .and. (Q1 .lt. 45)) .or. (Q1 .eq. 46) 
* .or. ((Q1 .gt. 57) .and. (Q1 .lt. 64)) .or. (Q1 .eq. 92) 
* .or. (Q1 .eq. 93) .or. (Q1 .eq. 124)) then
C (invalid input)
call BEEP(FLDROM,FLDCOL+H)
call ERRMSG(17,1)
else
H=H+1
if ((Q1 .gt. 96) .and. (Q1 .lt. 123)) then
Q(H)=char(Q1-32)
else
Q(H)=char(Q1)
endif
call MRCHAT(PAGE,Q(H),ATRIB3,1)
if (H .eq. FIELD) then
(field full)
call CSROFF
call PRINT(PAGE,FLDROM,FLDCOL+FIELD+1,'<--J',ATRIB1+128,3)
endif
endif
go to 300
400 DRIVE='B:'
do 30 J=1,8
if (Q(J) .eq. ' ') goto 500
TEMP(J+2)=Q(J)
30 continue
do 50 K=1,4
TEMP(J+K+1)=EXT(K)
50 continue
if (TODISK) then
if (EXT(2) .eq. 'L') then
call SAVE(FILSPC)
else
call DISBOL(FILSPC)
endif
endif
inquire(file=FILSPC,exist=OKFILE)
if (.not. OKFILE) then
call BEEP(FLDROM,FLDCOL+H)
call ERRMSG(17,3)
call CSRON
goto 100
endif
if (EXT(2) .eq. 'L') then
call LOAD(FILSPC)
else
call BOLDIS(FILSPC)
endif
NODATA=.false.
NENDAT=.true.
endif
call CSROFF
return
end
SUBROUTINE HELP1(CHOICE)
$INCLUDE:'QATRIB.BLK'

integer*2 CHOICE
if (CHOICE .eq. 1) then
  call PRINT(0,21,21,' Edit a Ladder Diagram File ', *ATRIB9,40)
elseif (CHOICE .eq. 2) then
  call PRINT(0,21,21,' Load a Boolean or Ladder File from Disk ', *ATRIB9,40)
elseif (CHOICE .eq. 3) then
  call PRINT(0,21,21,' Save a Boolean or Ladder File to Disk ', *ATRIB9,40)
elseif (CHOICE .eq. 4) then
  call PRINT(0,21,21,' Upload a Boolean Program from Controller ', *ATRIB9,40)
elseif (CHOICE .eq. 5) then
  call PRINT(0,21,21,' Download a Boolean Program to Controller ', *ATRIB9,40)
else
  call PRINT(0,21,21,' Return to Disk Operating System (DOS) ', *ATRIB9,40)
endif
return
end

CSUBROUTINE CRTTYP
C (select monochrome of color CRT)
$INCLUDE:'QATRIB.BLK'
integer*2 I, J
integer*2 BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW,
* WHITE, BBLACK, BBLUE, BGREEN, BCYAN, BRED,
* BMAGENTA, BYELLOW, BWHITE, BLINK, HIGHINT
data BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW /0,1,2,3,4,5,6/,
* WHITE, BBLACK, BBLUE, BGREEN, BCYAN, BRED /7,0,16,32,48,64/,
* BMAGENTA, BYELLOW, BWHITE, BLINK, HIGHINT /80,96,112,128,8/
ATRIB0=WHITE+BBLACK
ATRIB1=WHITE+BBLACK+HIGHINT
call CSROFF
call CLS(ATRIB0)
call PRINT(PAGE,8,20,'PROGRAMMABLE CONTROLLER PROGRAMING SYSTEM', *ATRIB0,41)
call PRINT(PAGE,11,31,'Copyright (C) 1985',ATRIB0,19)
call PRINT(PAGE,13,35,'Version 1.0',ATRIB0,11)
call PRINT(PAGE,15,31,'All Rights Reserved',ATRIB0,19)
call PRINT(PAGE,19,17,'(press ',ATRIB0,7)
call WRCAT(PAGE,'c',ATRIB1,1)
call PRINT(PAGE,19,25,' for color or ',ATRIB0,14)
call WRCAT(PAGE,'m',ATRIB1,1)
call PRINT(PAGE,19,40,' for monochrome display)',ATRIB0,24)
100 call INKEY(I,J)
if ((I .eq. ichar('c')) .or. (I .eq. ichar('C'))) then
  (color attribute set)
  ATRIB0 = WHITE + BBLUE + BLINK
  ATRIB1 = YELLOW + BBLACK
  ATRIB2 = BLACK + BYELLOW
  ATRIB3 = WHITE + BBLACK
  ATRIB4 = BLACK + BWHITE
  ATRIB5 = BLACK + BCYAN
  ATRIB6 = WHITE + BBLUE
  ATRIB7 = WHITE + BRED
  ATRIB8 = CYAN + BBLACK
  ATRIB9 = BLACK + BCYAN
Appendix F. SYSTEM PROGRAM SOURCE LISTING 160
```c
elseif ((I .eq. ichar('m')) .or. (I .eq. ichar('M'))) then

   ATRIB0 = BLACK + BWHITE + BLINK
   ATRIB1 = WHITE + BBLACK
   ATRIB2 = BLACK + BWHITE
   ATRIB3 = WHITE + BBLACK
   ATRIB4 = BLACK + BWHITE
   ATRIB5 = WHITE + BBLACK
   ATRIB6 = BLACK + BWHITE
   ATRIB7 = BLACK + BWHITE
   ATRIB8 = WHITE + BBLACK
   ATRIB9 = BLACK + BWHITE

else
   call BEEP(1,1)
   goto 100
endif
return
end

SUBROUTINE INITLL

$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

integer*2 I, J

ENDLDR=.false.
ELFULL=.false.
ELFREE=1
ELHEAD=1
ELETOT=0
do 10 J=1,ELEMAX
   ELMNTI(J,1)=J-1
   ELMNTI(J,2)=J+1
   ELMNTI(J,3)=0
10 continue
ELMNTI(ELEMAX,2)=-1

NWFULL=.false.
NWFREE=1
NWFHEAD=1
NWTAIL=1
NTHPTR=1
NTHTOT=0
NTHNUM=1
ABSRNG=1
do 20 J=1,NTWMAX
   NETWRK(J,4)=J-1
   NETWRK(J,5)=J+1
20 continue
NETWRK(NTWMAX,5)=-1
do 30 J=1,1023
   TYPREF(J)= ' ' 
30 continue
SBXTOT=0
do 50 I=1,128
   SBXREF(I)=0
   SBOXC(I)= ' ',
do 40 J=1,3
   SBOX(I,J)=0
40 continue
SBXTOT=0
do 50 I=1,130
   LBOXREF(I)=0
   LBOXC(I)= ' ',
do 60 J=1,50
   LBOX(I,J)=0
60 continue
```

Appendix F. SYSTEM PROGRAM SOURCE LISTING
SUBROUTINE ERRMSG(RON, MSG)

$Iinclude: 'QATRIB.BLK'$

integer*2 MSG, Q1, Q2, RON

goto (1,2,3,4,5,6,7,8) MSG
return
1 call PRINT(PAGE, RON, 35, ' Invalid Key ', ATRIB7, 13)
goto 100
2 call PRINT(PAGE, RON, 25, ' Option Not Currently Available ', ATRIB7, #52)
goto 100
3 call PRINT(PAGE, RON, 31, ' File Does Not Exist ', ATRIB7, 21)
goto 100
4 call PRINT(PAGE, RON, 15, ' Enter File Name or Press F1 To Return To
Main Menu ', ATRIB7, #52)
goto 100
5 call PRINT(PAGE, RON, 16, ' Input Field Full - Press J To Enter Fil
name ', ATRIB7, #46)
goto 100
6 call PRINT(PAGE, RON, 25, ' There Is Not A File In Memory ', ATRIB7, 31
*)
goto 100
7 call PRINT(PAGE, RON, 20, ' Requested File Is Not A Ladder Data File
* ', ATRIB7, #42)
goto 100
8 call PRINT(PAGE, RON, 25, ' Error Reading Ladder Data File ', ATRIB7, 3
*)
goto 100
100 call INKEY(Q1, Q2)
call LOCATE(PAGE, RON, 5)
call HRCHAT(PAGE, ' ', ATRIB1, 72)
return
end

SUBROUTINE PRINT(PAGE, RON, COL, STRING, ATTRIB, LENGTH)

** WRITE A CHARACTER STRING TO THE SCREEN **

PAGE page on which to write
RON row on which to write string
COL column in which to begin writing string
STRING character string
LENGTH length of character string
ATTRIB screen attributes of string space

integer*2 ATTRIB, COL, I, LENGTH, PAGE, RON
character#1 STRING(LENGTH)
call LOCATE(PAGE, RON, COL)
do 10 I=1, LENGTH
call HRCHAT(PAGE, STRING(I), ATTRIB, 1)
10 continue
return
end
SUBROUTINE BEEP(RON, COLUMN)
$INCLUDE:'QATRIB.BLK'
integer*2 COLUMN, RON

call CSROFF
call LOCATE(PAGE,1,1)
write(*,*) char(7)
call LOCATE(PAGE,ROW,COLUMN)
return
end

SUBROUTINE EDITOR(NODATA, NENDAT)
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

integer*2 I, Q1, Q2
logical*2 NENDAT, NODATA

call CSROFF
PAGE=2
call SETPAG(2)
if (NODATA) then
  NODATA=.false.
  NTNTOT=1
  NAFREE=2
  NETKRI(1,5)=-1
  NETKRI(2,4)=0
  ENLDLR=.true.
call PRIN(PAGE,25,69,' 1adder end ',ATRIB7,12)
call INITNM
else
  if (NENDAT) call SHO
endif
NENDAT=.false.

 TABLE OF KEY FUNCTIONS
 cursor up - move cursor to element above the current element or move to data input field above the current field
 cursor down - move cursor to element below the current element or move to data input field below the current field
 cursor left - move to element to the left of the current element or move to data input field to the left of current field
 cursor right - move to element to the right of the current element or move to data input field to the right of current field
 PgUp - move cursor to rung one in current column
 PgDn - move cursor to rung seven in current column
 Home - move cursor to column one on current rung
 End - move cursor to column twelve on current rung
 Ctrl-PgUp - display the previous ladder network
 Ctrl-PgDn - display the next ladder network
 Ctrl-Home - display the first ladder network
 Ctrl-End - display the last ladder network
 ESC - escape from (undo) last command or erase keyboard input before it is entered
 carriage return - move to beginning of next row below or enter the keyboard input
 tab - same as cursor right but two cursor positions at a time
 back tab - same as cursor left but two cursor positions at a time
 insert - insert a row above or a column to the left of the current cursor position, or a network above the current network

Appendix F. SYSTEM PROGRAM SOURCE LISTING 163
100 call INKEY(Q1,Q2)
    if (Q1 .eq. 0) then
        C {EXTENDED CODE KEYS}
        C {function keys F1-F10)
        goto (1,2,3,4,5,6,7,8,9,10) Q2-58
        goto 200
    1 return
    goto 100
    2 call EDIT
    goto 100
    3 call SMBOX
    goto 100
    4 call LGBOX
    goto 100
    5 call VERTCL
    goto 100
    6 call HORIZ
    goto 100
    7 call CONTCT(Q2)
    goto 100
    8 call COIL(Q2)
    goto 100
    9 call CONTCT(Q2)
    goto 100
    10 call COIL(Q2)
    goto 100
200 if (Q2 .eq. 72) then
    C {cursor up}
    call MOVEUP
    elseif (Q2 .eq. 80) then
    C {cursor down}
    call MOVEDN
    elseif (Q2 .eq. 75) then
    C {cursor left)
    call MOVLFT
    elseif (Q2 .eq. 77) then
    C {cursor right}
    call MOVRGT
    elseif (Q2 .eq. 71) then
    C {home)
    call TOHOME
    elseif (Q2 .eq. 79) then
    C {end)
    call TOEND
    elseif (Q2 .eq. 119) then
    C {ctrl-home}
    call CTLHOM
    elseif (Q2 .eq. 117) then
    C {ctrl-end)
    call CTLEND
    elseif (Q2 .eq. 73) then
    C {PgUp)
    call PAGEUP
    elseif (Q2 .eq. 81) then
    C {PgDn)
    call PAGEDN
    elseif (Q2 .eq. 132) then
    C {Ctrl-PgUp)
    call CPSUP
    elseif (Q2 .eq. 118) then
    C {Ctrl-PgDn)
    call CPGDN
    endif
elseif (Q2 .eq. 82) then
  (insert)
call INSERT
elseif (Q2 .eq. 83) then
  (delete)
call DELETE
else
  call BEEP(2,6)
endif
C (ASCII KEYS)
else
if (Q1 .eq. 13) then
  (carriage return)
call HILITE(ATRIB1)
COLUMN=1
if (RUNG .eq. 7) then
  RUNG=1
else
  RUNG=RUNG+1
endif
call HILITE(ATRIB4)
call COORD
else
  call BEEP(2,6)
endif
endif
goto 100
end

SUBROUTINE MOVEUP
C (move cursor up one element position)
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
character*1 ELETYP
C ELETYP - element type at the current cursor position (temporary var)
if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(48)) then
    (check if current element is a large box)
call BEEP(ROW(RUNG),COL(COLUMN))
return
endif
call HILITE(ATRIB1)
if (RUNG .eq. 1) then
  (wrap around to bottom)
  RUNG=7
else
  RUNG=RUNG-1
endif
if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  (not a null element)
  ELETYP=ELMNTC(ELEPTR(RUNG,COLUMN),1)
  if (ELETYP .ge. char(21)) then
    (small box)
    RUNG=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),2))
    COLUMN=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),3))
  endif
call HILITE(ATRIB4)
call COORD
return
end
SUBROUTINE MOVEDN
C (move cursor down one element position)

$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QLADDER.BLK'

character*1 ELETYP

if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(48)) then
    (check if current element is a large box)
    call BEEP(ROM(RUNG),COL(COLUMN))
    return
  endif
endif
call HILITE1
if (RUNG .eq. 7) then  
  (wrap around to top)
  RUNG=1
else
  if (ELEPTR(RUNG,COLUMN) .le. 0) then
    (null element)
    RUNG=RUNG+1
  else
    ELETYP=ELMNTC(ELEPTR(RUNG,COLUMN),1)
    if (ELETYP .lt. char(21)) then
      (contact,coil,horizontal or vertical)
      RUNG=RUNG+1
    else
      (small box)
      RUNG=RUNG+3
    endif
  endif
endif
if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  if (ELETYP .ge. char(21)) then
    (small box)
    COLUMN=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),3))
  endif
endif
call HILITE4
return
end

SUBROUTINE MOVLFT
C (move cursor left one element position)

$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QLADDER.BLK'

character*1 ELETYP

call HILITE1
if (COLUMN .eq. 1) then
  (wrap around to right side)
  COLUMN=12
else
  COLUMN=COLUMN-1
endif
if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  ELETYP=ELMNTC(ELEPTR(RUNG,COLUMN),1)
  if (ELETYP .ge. char(21)) then
    (small box)
    COLUMN=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),3))
  endif
call HILITE4
call COORD
return
end
SUBROUTINE MOVRGT
C {move cursor right one element position}
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
character*1 ELETYP

call HILITE(ATTRIB4)
if (COLUMN .eq. 12) then
   COLUMN=1
else
   if (ELEPTR(RUNG,COLUMN) .le. 0) then
      COLUMN=COLUMN+1
   else
      ELETYP=ELMNT(ELEPTR(RUNG,COLUMN),1)
      if (ELETYP .lt. char(21)) then
         COLUMN=COLUMN+1
      elseif (ELETYP .lt. char(48)) then
         COLUMN=COLUMN+5
      else
         COLUMN=12
      endif
   endif
endif
if (ELEPTR(RUNG,COLUMN) .gt. 0) then
   ELETYP=ELMNT(ELEPTR(RUNG,COLUMN),1)
   if (ELETYP .ge. char(21)) then
      RUNG=ichar(ELMNT(ELEPTR(RUNG,COLUMN),2))
   endif
endif
call HILITE(ATTRIB4)
call COORD
return

SUBROUTINE TOHOME
C {move cursor to left side of screen - network column 1}
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
if (COLUMN .eq. 1) return
call HILITE(ATTRIB1)
COLUMN=1
call HILITE(ATTRIB4)
call COORD
return
SUBROUTINE TOEND

C (move cursor to right side of screen - network column 12)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

if (COLUMN .eq. 12) return
call HILITE(ATRIB1)
COLUMN=12
call HILITE(ATRIB4)
call COORD
return
end

SUBROUTINE CTLHOM

C (move cursor to rung 1, column 1 in first network of ladder file)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

if (NTNPTR .eq. NNHEAD) then
    call BEEP(ROH(RUNG),COL(COLUMN))
else
    NTNPTR=NNHEAD
    NTNUM=1
    RUNG=1
    COLUMN=1
    ABSRNG=1
    if (ENDLDR) then
        ENDLDR=.false.
        call LOCATE(PAGE,25,69)
        call HRCHAT(PAGE,' ',ATRIBS,12)
    endif
    call COORD
    call SHONNN
endif
return
end

SUBROUTINE CTLEND

C (move cursor to rung 1, column 1 in last network of ladder file)

$INCLUDE:'QLADDER.BLK'

if (NTNPTR .eq. NNTAIL) then
    call BEEP(ROH(RUNG),COL(COLUMN))
else
    NTNUM=NTTOT
    RUNG=1
    COLUMN=1
    ENDLDR=.true.
    ABSRNG=1
    (count number of rungs in ladder up to last network)
    NTNPTR=NNHEAD
100 ABSRNG=ABSRNG+NETWK(NTNPTR,3)
    NTNPTR=NETWK(NTNPTR,5)
    if (NTNPTR .ne. NNTAIL) goto 100
    call COORD
    call SHONNN
endif
return
end

Appendix F. SYSTEM PROGRAM SOURCE LISTING
SUBROUTINE PAGEUP
C (move cursor to top row of network - network rung 1)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 ELETYP

if (RUNG .eq. 1) return

if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  ELETYP=ELMNTC(ELEPTR(RUNG,COLUMN),1)
  if (ELETYP .ge. char(21)) then
    COLUMN=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),3))
  endif
endif

call HILITE(ATRIB1)
call COORD
return
der

SUBROUTINE PAGEDOWN
C (move cursor to bottom row of network - network rung 7)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

if (RUNG .eq. 7) return

if (ELEPTR(RUNG,COLUMN) .gt. 0) then
  if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(48)) return
endif

ELETYP=ELMNTC(ELEPTR(RUNG,COLUMN),1)
if (ELETYP .ge. char(21)) then
  COLUMN=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),3))
endif

RUNG=7

call HILITE(ATRIB1)
call COORD
return
der

SUBROUTINE CPUP
C (move cursor to top row of preceding network - network rung 1)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

if (NTNTR .eq. NMHEAD) then
  call BEEP(ROW(RUNG),COL(COLUMN))
else
  CALL BEEP(ROW(RUNG),COL(COLUMN))
  NTHNUM=NTNTR
  NTHNUM=NTNTR
  NTHNUM=NTNTR
  NTHNUM=NTNTR
  RUNG=1
  COLUMN=1
  ABSRNG=ABSRNG-NETWK(NTNTR,3)
if (ENDLDR) then
  ENDLDR=.false.
call LOCATE(PAGE,25,69)
call WRCHAT(PAGE,' ',ATRIB5,12)
endif

call COORD
call SHOWN
endif
return
end

C

SUBROUTINE CPGDN
C (move cursor to top row of following network - network rung 1)

$INCLUDE:'QLADDER.BLK'

integer*2 NEWREC

NTNUM=NTNUM+1
ABSRNG=ABSRNG+NETHRK(NTHPTR,3)
if (NTHPTR .eq. NTAIL) then
  (add a new network at end of ladder)
  if (NMFULL) then
    (network list is full)
    call BEEP(2,6)
    return
  else
    NTNTOT=NTNTOT+1
    NENREC=NNFREE
    (set unused record ptr to next unused record)
    NNFREE=NETHRK(NNFREE,5)
    (set next unused record's bkwd ptr to 0)
    NETHRK(NNFREE,4)=0
    (check if linked list is full)
    if (NNFREE .eq. -1) NMFULL=.true.
    (set new record's bkwd ptr to preceding record)
    NETHRK(NEWREC,4)=NTHPTR
    (set new record's fwd ptr to indicate it is last record)
    NETHRK(NEWREC,5)=-1
    (set network list tail ptr to new record)
    NNTAIL=NEWREC
    (set preceding record's fwd ptr to new record)
    NETHRK(NTHPTR,5)=NEWREC
    (set current record ptr to new record)
    NTNPTR=NEWREC
    (initialize new network)
call INITNN
  endif
else
  NTNPTR=NETHRK(NTNPTR,5)
  RUNG=1
  COLUMN=1
call COORD
call SHOWN
endif
return
end

C

SUBROUTINE HILITE(ATRIB)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 CHARR, DUMMY, ELETYP
integer*2 ATRIB, COL1, COL2, I, J, ROW1, ROM2
if (ELEPTR(RUNG, COLUMN) .le. 0) then
  if (COLUMN .eq. 12) then
    call PRINT(PAGE, ROW(RUNG), COL(COLUMN), ', ', ATTRIB, 6)
    call PRINT(PAGE, ROW(RUNG) + 1, COL(COLUMN), ', ', ATTRIB, 6)
    call LOCATE(PAGE, ROW(RUNG), COL(COLUMN))
  else
    do 20 I = ROW(RUNG), ROW(RUNG) + 1
       do 10 J = COL(COLUMN), COL(COLUMN) + 5
          call LOCATE(PAGE, I, J)
          call RDCHAT(PAGE, CHAR, DUMMY)
          call NRCHAT(PAGE, CHAR, ATTRIB, 1)
       10 continue
    20 continue
  endif
endif
ELETYP = ELNTC(ELEPTR(RUNG, COLUMN), 1)
if (ELETYP .lt. char(21)) then
  RON1 = ROW(RUNG)
  RON2 = RON1 + 1
  COL1 = COL(COLUMN)
  COL2 = COL1 + 5
elseif (ELETYP .lt. char(48)) then
  RON1 = ROW(RUNG)
  RON2 = RON1 + 8
  COL1 = COL(COLUMN) + 1
  COL2 = COL1 + 14
else
  RUNG = 1
  COLUMN = 3
  RON1 = ROW(RUNG) - 1
  RON2 = RON1 + 21
  COL1 = COL(COLUMN) + 1
  COL2 = COL1 + 52
endif
  do 40 I = RON1, RON2
     do 30 J = COL1, COL2
        call LOCATE(PAGE, I, J)
        call RDCHAT(PAGE, CHAR, DUMMY)
        call NRCHAT(PAGE, CHAR, ATTRIB, 1)
     30 continue
  40 continue
return
end

SUBROUTINE INITNW

{initializes a network to have one rung with one NO contact at
 position (1, 1), one NO output at position (1, 12) and horizontal
 elements in between}
{NOTE: set NTMPTR before calling this subroutine}

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
character*1 C(5)
integer*2 I, J

{increase total element counter}
ELETOT = ELETOT + 12

{check if element data list is full}
if (ELETOT .eq. ELEMAX) then
  ELFULL = true.
  call PRINT(PAGE, 25, 56,' ', 'ladder full ', ATRIB7, 13)
  endif

{check if element data list has room for another network}

Appendix F. SYSTEM PROGRAM SOURCE LISTING 171
if (ELEMAX-ELETOT .lt. 12) NNFULL=.true.
C (set first element's cell reference ptr to first free record)
ELEPTR(1,1)=ELFREE
C (set first element of network ptr to first element ptr)
NETNRK(NTNPTR,1)=ELFREE
C (set the cell reference ptrs of other elements on rung 1)
do 10 J=2,12
   ELEPTR(1,J)=ELENTI(ELEPTR(1,J-1),2)
10 continue
C (set last element of network ptr to last element ptr)
NETNRK(NTNPTR,2)=ELEPTR(1,12)
C (set number of rungs in network to 1)
NETNRK(NTNPTR,3)=1
C (set unused record ptr to next unused record)
ELFREE=ELENTI(ELEPTR(1,12),2)
C (set bkwd ptr of first unused record to 0)
ELMNTI(ELFREE,1)=0
C (set the cell reference pointers of all null element screen cells)
do 20 I=2,7
   do 30 J=1,12
      ELEPTR(I,J)=ELEPTR(I,11)
   30 continue
   ELEPTR(I,12)=-ELEPTR(I,12)
20 continue
C (set lst element's bkwd ptr & preceding element's fwd ptr)
if (NTNPTR .eq. NNHEAD) then
  (first network in ladder)
  ELENTI(ELEPTR(1,1),1)=0
  ELHEAD=ELEPTR(1,1)
else
  ELENTI(ELEPTR(1,1),1)=NETNRK(NTNPTR,4,2)
  ELENTI(NETNRK(NTNPTR,4,2),2)=ELEPTR(1,1)
endif
C (set other elements' bkwd ptrs to previous element)
do 40 J=2,12
   ELENTI(ELEPTR(1,J),1)=ELEPTR(1,J-1)
40 continue
C (set other elements' fwd ptrs to following element)
do 50 J=1,11
   ELENTI(ELEPTR(1,J),2)=ELEPTR(1,J+1)
50 continue
C (set 12th element's fwd ptr & following element's bkwd ptr)
if (NTNPTR .eq. NNTAIL) then
  (last network in ladder)
  ELENTI(ELEPTR(1,12),2)=1
else
  ELENTI(ELEPTR(1,12),2)=NETNRK(NTNPTR,5,1)
  ELENTI(NETNRK(NTNPTR,5,1),1)=ELEPTR(1,12)
endif
C (enter element types in data list)
ELENTC(ELEPTR(1,1),1)=char(3)
ELENTC(ELEPTR(1,12),1)=char(10)
do 60 I=2,12
   ELENTC(ELEPTR(1,I),1)=char(1)
60 continue
C (enter element row & column in data list)
do 70 I=1,12
   ELENTC(ELEPTR(1,I),2)=char(1)
   ELENTC(ELEPTR(1,I),3)=char(1)
70 continue
C (enter element link data in data list)
do 80 I=1,7
   ELENTC(ELEPTR(1,I),4)=char(3)
   ELENTC(ELEPTR(1,I),4)=char(48)
80 continue
do 90 J=1,7
   ROWCNT(J)=0
90 continue
90 continue
C {display initial elements on screen}
call DONSCL(0,1,22,1,4,ATRIB1)
call DONSCL(0,1,22,6,79,ATRIB1)
do 15 J=1,22
   call LOCATE(PAGE,J,5)
call WRCHAT(PAGE,'|',ATRIB1,1)
call LOCATE(PAGE,J,79)
call WRCHAT(PAGE,'|',ATRIB1,1)
15 continue

call BINCRL(NUM,C(1),C(2),C(3),C(4),C(5))
call LOCATE(PAGE,25,10)
do 25 J=2,5
   if (C(J) .eq. char(0)) C(J)='0'
call WRCHAT(PAGE,C(J),ATRIB5,1)
25 continue

RUNG=1
do 35 J=1,12
   COLUMN=J
   ROWCNT(1)=ROWCNT(1)+1
call NRELEM(ATRIB1)
35 continue

COLUMN=1
call COORD
call HILITE(ATRIB4)
return
c

SUBROUTINE COORD
C display current cursor coordinates on message line
$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QLADDER.BLK'
c
character*1 C(5)
call BINCRL(RUNG,C(1),C(2),C(3),C(4),C(5))
call LOCATE(PAGE,25,20)
call WRCHAT(PAGE,C(1),ATRIB5,1)
call BRANCH(COLUMN,C(1),C(2),C(3),C(4),C(5))
call LOCATE(PAGE,25,30)
call WRCHAT(PAGE,C(4),ATRIB5,1)
call WRCHAT(PAGE,C(5),ATRIB5,1)
return
c
SUBROUTINE INSERT
C {insert a row above, a column or element to the left of the current
C cursor position or a network above the current network}
$INCLUDE: 'QLADDER.BLK'
c
integer*2 Q1, Q2
call PROMPT(9)
100 call INKEY(Q1,Q2)
   if (Q1 .eq. 0) then
      if (Q2 .eq. 60) then
call INSNTN
c   elseif (Q2 .eq. 64) then
call INSCOL
c   elseif (Q2 .eq. 63) then
call INSELE
c   elseif (Q2 .eq. 68) then
call INSROM
celse
call BEEP(2,6)
goto 100
endif
else
  (esc)
    if (Q1 .ne. 27) then
      call BEEP(2,6)
goto 100
    endif
call PROMPT(KEY)
return
end

SUBROUTINE INSNTN
C (insert a network above the current network)
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLadder.BLK'

integer*2 NEWREC

if (NMFULL) then
  (network list is full)
call BEEP(2,6)
return
else
  NTNTOT=NTNTOT+1
  NEWREC=NNFREE
  (set unused record ptr to next unused record)
  NMFREE=NETWRK(NMFREE,5)
  (set next unused record's bkwd ptr to 0)
  NETWRK(NMFREE,4)=0
  (check if linked list is full)
  if (NMFREE .eq. -1) NMFULL=.true.
  (set new record's bkwd ptr to preceding record)
  NETWRK(NEWREC,4)=NETWRK(NTNPTR,4)
  (set new record's fwd ptr to current record)
  NETWRK(NEWREC,5)=NTNPTR
  (check if new record is first network in ladder)
  if (NETWRK(NEWREC,4).eq. 0) then
    NMHEAD=NEWREC
  else
    (set preceding record's fwd ptr to new record)
    NETWRK(NETWRK(NEWREC,4),5)=NEWREC
  endif
  (set current record's bkwd ptr to new record)
  NETWRK(NTNPTR,4)=NEWREC
  (set current record ptr to new record)
  NTNPTR=NEWREC
  (initialize new network)
  if (ENDLDR) then
    ENDLR=.false.
call LOCATE(PAGE,25,69)
call NRCHAT(PAGE,' >ATRIB5',12)
  endif
call INITNN
return
end

SUBROUTINE WRELEM(ATTRIB)
C (NOTE: set RUNG and COLUMN before calling this subroutine)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLadder.BLK'
integer*2 ATTRIB, COUNT, ELETYP, I, J

ELETYP=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),1))
if (ELETYP .eq. 2) then
    call HRLNKL(RUNG,COLUMN,ATTRIB1)
    call HRLNKR(RUNG,COLUMN,ATTRIB)
    return
C --- contact or horizontal ---
elseif (ELETYP .eq. 9) then
    call LOCATE(PAGE,RUNG+1,COLUMN)
    if ((ELETYP .eq. 3) .or. (ELETYP .eq. 6)) then
        call MRCHAT(PAGE,'X',ATTRIB,1)
        call MRCHAT(PAGE,' ',ATTRIB,4)
    elseif ((ELETYP .eq. 4) .or. (ELETYP .eq. 7)) then
        call MRCHAT(PAGE,'Y',ATTRIB,1)
        call MRCHAT(PAGE,' ',ATTRIB,4)
    elseif ((ELETYP .eq. 5) .or. (ELETYP .eq. 8)) then
        call MRCHAT(PAGE,'C',ATTRIB,1)
        call MRCHAT(PAGE,' ',ATTRIB,4)
    else
        call MRCHAT(PAGE,'°',ATTRIB,5)
    endif
    if (COLUMN .eq. 1) then
        call LOCATE(PAGE,RUNG+1,COLUMN+3)
        call MRCHAT(PAGE,'Y',ATTRIB,1)
        call MRCHAT(PAGE,' ',ATTRIB,4)
    endif
    if (ELETYP .eq. 1) then
        call LOCATE(PAGE,RUNG+1,COLUMN+1)
        call MRCHAT(PAGE,'—',ATTRIB,5)
    elseif (ELETYP .eq. 6) then
        call PRINT(PAGE,RUNG+1,COLUMN,'—Ä—',ATTRIB,5)
    else
        call BEEPIRON(RUNG),COLUMN)
    endif
    call LOCATE(PAGE,RUNG+1,79)
    call MRCHAT(PAGE,'F',ATTRIB,1)
C -------- small box --------
elseif (ELETYP .eq. 48) then
    call LOCATE(PAGE,RUNG+1,COLUMN)
    call MRCHAT(PAGE,'°',ATTRIB,1)
    call MRCHAT(PAGE,'',ATTRIB,1)
    call MRCHAT(PAGE,'°',ATTRIB,13)
    call LOCATE(PAGE,RUNG+1,COLUMN+2+3)
call WRCHAT(PAGE,' ',ATTRIB,1)
do 10 J=1,7
  call LOCATE(PAGE,ROW(RUNG)+J,COL(COLUMN))
call WRCHAT(PAGE,' ',ATTRIB,1)
call WRCHAT(PAGE,'\',ATTRIB,13)
call LOCATE(PAGE,ROW(RUNG)+J,COL(COLUMN+2)+3)
call WRCHAT(PAGE,' ',ATTRIB,1)
if ((J .eq. 4) .or. (J .eq. 7)) then
call WRCHAT(PAGE,' ',ATTRIB,1)
else
call WRCHAT(PAGE,' ',ATTRIB,2)
endif
10 continue
call LOCATE(PAGE,ROW(RUNG)+8,COL(COLUMN)+1)
call WRCHAT(PAGE,'\',ATTRIB,1)
call WRCHAT(PAGE,\',',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+8,COL(COLUMN+2)+3)
call WRCHAT(PAGE,'\',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN))
call WRCHAT(PAGE,' ',ATTRIB,1)
call WRCHAT(PAGE,' ',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN+2)+3)
call WRCHAT(PAGE,' ',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN))
call WRCHAT(PAGE,\',',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)+1)
call WRCHAT(PAGE,\',',ATTRIB,1)
call WRCHAT(PAGE,\',',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)+1)
call WRCHAT(PAGE,\',',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)+1)
call WRCHAT(PAGE,\',',ATTRIB,1)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)+1)
call WRCHAT(PAGE,\',',ATTRIB,1)
if (ELETYP .ge. 41) then
  (small two-input box)
call LOCATE(PAGE,ROW(RUNG)+1+1,COL(COLUMN)-1)
call WRCHAT(PAGE,'-',ATTRIB,2)
call LOCATE(PAGE,ROW(RUNG)+1+1,COL(COLUMN)+1)
call WRCHAT(PAGE,\',',ATTRIB,1)
endif
if (ELETYP .ge. 45) then
  (small three-input box)
call LOCATE(PAGE,ROW(RUNG)+2+1,COL(COLUMN)-1)
call WRCHAT(PAGE,'-',ATTRIB,2)
call LOCATE(PAGE,ROW(RUNG)+2+1,COL(COLUMN)+1)
call WRCHAT(PAGE,\',',ATTRIB,1)
endif
if (ELETYP .eq. 38) then
call PRINT(PAGE,ROW(RUNG)+1,\',',ATTRIB,8)
elseif (ELETYP .eq. 41) then
call PRINT(PAGE,3,COL(COLUMN)+3,'ONE SHOT',\',',ATTRIB,8)
elseif (ELETYP .eq. 42) then
call PRINT(PAGE,3,COL(COLUMN)+3,'COUNTER',\',',ATTRIB,7)
elseif (ELETYP .eq. 47) then
call PRINT(PAGE,3,COL(COLUMN)+3,'UP/DOWN',\',',ATTRIB,7)
elseif (ELETYP .eq. 43) then
call PRINT(PAGE,4,COL(COLUMN)+3,'COUNTER',\',',ATTRIB,10)
endif
call PRINT(PAGE,6,COL(COLUMN)+3,'PRESET =',\',',ATTRIB,8)
call WRLNKR(RUNG,COLUMN+2,ATTRIB)
call DNSCRNL(PAGE,5,21,6,17,ATTRIB1)
do 20 J=6,21
  call LOCATE(PAGE,J,5)
call WRCHAT(PAGE,\',',ATTRIB,1)
20 continue
call LOCATE(PAGE,1,19)
call WRCHAT(PAGE,'\',ATTRIB,1)
call WRCHAT(PAGE,'-',ATTRIB,51)
call LOCATE(PAGE,1,71)
call WRCHAT(PAGE,'\',ATTRIB,1)
do 30 J=1,20
  call LOCATE(PAGE,1+J,18)
call WRCHAT(PAGE,\',',ATTRIB,1)
call WRCHAT(PAGE,\',',ATTRIB,1)
call WRCHAT(PAGE,' ',ATTRIB,51)
call LOCATE(PAGE,1,J,71)
call WRCHAT(PAGE,' |',ATTRIB,1)
30 continue

call LOCATE(PAGE,22,19)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,5,39)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,5,71)
call WRCHAT(PAGE,' |',ATTRIB,1)
call PRINT(PAGE,2,21,'DRUM',ATTRIB,4)
call PRINT(PAGE,3,21,'RESET STEP =',ATTRIB,12)
call PRINT(PAGE,4,21,'SEC/COUNT =',ATTRIB,11)
call PRINT(PAGE,5,21,'STEP COUNT/STEP=',ATTRIB,16)
call LOCATE(PAGE,6,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,7,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,8,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,9,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,10,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,11,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,12,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,13,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call LOCATE(PAGE,14,23)
call WRCHAT(PAGE,' |',ATTRIB,1)
call PRINT(PAGE,15,22,'10',ATTRIB,2)
call PRINT(PAGE,16,22,'11',ATTRIB,2)
call PRINT(PAGE,17,22,'12',ATTRIB,2)
call PRINT(PAGE,18,22,'13',ATTRIB,2)
call PRINT(PAGE,19,22,'14',ATTRIB,2)
call PRINT(PAGE,20,22,'15',ATTRIB,2)
call PRINT(PAGE,21,22,'16',ATTRIB,2)
endif
if (ELETYP .lt. 48) call WRNLKLRUNG,COLUMN,ATTRIB1)
SUBROUTINE RMVELE
C (remove an element's record from the element linked list )
C (and set the cell reference pointers for the element to 0)

$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

integer=2 C, ELEMNT, J, R, TYPE

ELEMNT=ELEPTR(RUNG,COLUMN)
TYPE=ichar(ELEMNT(ELEMNT,1))
ELETOT=ELETOT-1
if (NNFULL) then
  if ((ELEMNT(ELEMNT,1).eq.12).and.(NNFREE.ne.1)) NNFULL=.false.
endif

C (check if removed element was first element in network)
if (ELEMNT.eq.NETNRK(NTNPTR,1)) then
  C (set first element in network ptr to following element)
  NETNRK(NTNPTR,1)=ELEMNT(ELEPTR(RUNG,COLUMN),2)
endif
C (check if removed element was last element in network)
if (ELEMNT.eq.NETNRK(NTNPTR,2)) then
  C (set last element in network ptr to preceding element)
  NETNRK(NTNPTR,2)=ELEMNT(ELEPTR(RUNG,COLUMN),1)
endif
C (check if removed element was first element in ladder)
if (ELEMNT.eq.ELHEAD) then
  C (set following record's bkwd ptr to 0)
  ELEMNT(ELEMNT,2)=0
C (set first ladder element ptr to following element)
  ELHEAD=ELEMNT(ELEMNT,2)
C (check if removed element was last element in ladder)
elseif (ELEMNT(ELEMNT,2).eq.-1) then
  C (set preceding record's fwd ptr to indicate it is last record)
  ELEMNT(ELEMNT(ELEMNT,1),2)=-1
else
  C (set preceding record's fwd ptr to following record)
  ELEMNT(ELEMNT(ELEMNT,1),2)=ELEMNT(ELEMNT,2)
  C (set following record's bkwd ptr to preceding record)
  ELEMNT(ELEMNT(ELEMNT,2),1)=ELEMNT(ELEMNT,1)
endif
C (set removed record's bkwd ptr to 0)
ELEMNT(ELEMNT,1)=0
C (set removed record's fwd ptr to first unused record)
ELEMNT(ELEMNT,2)=ELFREE
C (set bkwd ptr of first unused record to removed record)
ELEMNT(ELFREE,1)=ELEMNT
C (set unused record pointer to removed record)
ELFREE=ELEMNT
C (set cell reference pointer(s) to 0 & initialize element data)
if (TYPE .lt. char(21)) then
  if (ELEMNT(ELEMNT,3).gt.0) then
    ELEMNT(ELEMNT,3)=0
    if ((TYPE .ne. 5) .and. (TYPE .ne. 8) .and. (TYPE .ne. 11) 
    .and. (TYPE .ne. 14)) then
      call BINASC(ABSRNG+COUNT,Q(1),Q(2),Q(3),Q(4),Q(5))
      call LOCATE(PAGE,ROW(I)+1,1)
      do 70 J=2,5
        if (Q(J) .eq. char(0)) Q(J)='0'
        call WRCHAT(PAGE,Q(J),ATRIB3,1)
      enddo
      COUNT=COUNT+1
    endif
  endif
  return
end
SUBROUTINE RNGDEC

C ( decrement number of elements on rung and update rung numbers )

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 Q(5)
integer*2 COUNT, I, J

RONCNT(RUNG)=RONCNT(RUNG)-1
if (RONCNT(RUNG) .lt. 1) then
C ( decrement number of rungs in network )
NETHRK(NTHPTR,3)=NETHRK(NTHPTR,3)-1
C ( erase rung number )
call LOCATE(PAGE,ROM(RUNG)+1,1)
call NRCHAT(PAGE,' ',ATRIB3,4)
C ( update rung numbers below empty rung )
COUNT=0
do 50 J=1,RUNG-1
   if (RONCNT(J) .gt. 0) COUNT=COUNT+1
50 continue
do 60 I=RUNG+1,7

C ----------—----———-——--———-——-—--——--——--———

end
if (RONCNT(I) .GT. 0) then
    call BINASC(ABSRNG+COUNT,Q(1),Q(2),Q(3),Q(4),Q(5))
    call LOCATE(PAGE,RON(I)+1,1)
    do 70 J=2,5
        if (Q(J) .eq. char(0)) Q(J)='0'
        call NRCHAT(PAGE,Q(J),ATRIB3,1)
    70 continue
    COUNT=COUNT+1
endif
60 continue
endif
return
end

SUBROUTINE DELETE
C (delete the row, column, or element at the current cursor position,
C or delete the current network, or the entire ladder)
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
integer*2 Q1, Q2
call PROMPT(10)
100 call INKEY(Q1,Q2)
    if (Q1 .eq. 0) then
        if (Q2 .eq. 60) then
            call DELNTN
        elseif (Q2 .eq. 63) then
            call DELELE
        elseif (Q2 .eq. 64) then
            call DELCOL
        elseif (Q2 .eq. 68) then
            call DELRON
        else
            call BEEP(2,6)
goto 100
        endif
    else
        C (esc)
        if (Q1 .eq. 27) then
            call BEEP(2,6)
goto 100
        endif
    endif
if (.not. ELFULL) then
    call LOCATE(PAGE,25,56)
call NRCHAT(PAGE,'I',ATRIB5,13)
endif
call PROMPT(KEY)
return
end

SUBROUTINE LGBOX
C (select a large box element and place at current cursor position)
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
character*1 TYPE
integer*2 Q1, Q2
if (ELFULL) then
    call BEEP(1,1)
    return
elseif (LBXTOT .gt. LBXMAX) then
SUBROUTINE SMBOX
C (select a small box element and place at current cursor position)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 TYPE
integer*2 COUNT, I, J, Q1, Q2

if (ELFULL) then
  call BEEP(1,1)
  return
elseif (SBXTOT .gt. SBXMAX) then
  call BEEP(1,1)
  return
elseif ((RUNG .ne. 1) or. (COLUMN .ne. 2) .and. (COLUMN .ne. 3)) then
  call BEEP(1,1)
  return
elseif (ELEPTR(1,3) .gt. 0) then
  if (ELMNTC(ELEPTR(1,3),1) .ge. char(48)) then
    call BEEP(1,1)
    return
  endif
endif
COUNT=0
do 10 I=1,3
  do 20 J=COLoNUM, COLUMN+2
    if (ELEPTR(I,J) .gt. 0) then
      if (ELMNTC(ELEPTR(I,J),1) .ge. char(21)) COUNT=COUNT+1
    endif
  20 continue
10 continue
  if ((COUNT .gt. 0) .and. (COUNT .lt. 6)) then
    call BEEP(2,6)
    return
  endif
100 call PROMPT(4)
200 call INKEY(Q1,Q2)
  if (Q1 .eq. 0) then
    if (Q2 .eq. 61) then
      C {counter type}
      call PROMPT(5)
      300 call INKEY(Q1,Q2)
        if (Q1 .eq. 0) then
          if (Q2 .eq. 61) then
            C {counter}
            TYPE=char(41)
            elseif (Q2 .eq. 62) then
              C {up/down counter}
              TYPE=char(47)
            else
              call BEEP(2,6)
              goto 300
            endif
          else
            C {esc}
            if (Q1 .eq. 27) then
              goto 100
            else
              call BEEP(2,6)
              goto 300
            endif
          endif
        elseif (Q2 .eq. 62) then
          C {timer}
          TYPE=char(42)
        elseif (Q2 .eq. 68) then
          C {up/down counter}
          TYPE=char(47)
        else
          call BEEP(2,6)
          goto 200
        endif
      else
        C {esc}
        if (Q1 .eq. 27) then
          call PROMPT(KEY)
          return
        else
          call BEEP(2,6)
          goto 200
        endif
      endif
    call ADDELE(TYPE)
    call WRELEM(ATRIB4)
    call EDIT
    call MOVRGT
    return
end

Appendix F. SYSTEM PROGRAM SOURCE LISTING 182
SUBROUTINE HORIZ
C place a horizontal connector at the current cursor position
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

if (ELFULL) then
    call BEEP(2,6)
    return
elseif (COLUMN .eq. 12) then
    call BEEP(1,1)
    return
endif
elseif (ELEPTR(RUNG,COLUMN) .gt. 0) then
    if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(21)) then
        call BEEP(1,1)
        return
    endif
endif
if (ELEPTR(1,3) .gt. 0) then
    if ((ELMNTC(ELEPTR(1,3),1).ge.char(48)) .and. (RUNG.ne.1)) then
        call BEEP(1,1)
        return
    endif
    elseif (ELMNTC(ELEPTR(1,3),1) .ge. char(45)) then
        continue
    elseif ((ELMNTC(ELEPTR(1,3),1) .ge. char(41)).and.(COLUMN .lt. 3)) then
        call BEEP(1,1)
        return
    endif
endif
endif
if (COLUMN .eq. 12) then
    call BEEP(1,1)
    return
elseif (ELFULL) then
    call BEEP(1,1)
    return
elseif (ELEPTR(RUNG,COLUMN) .gt. 0) then
    if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(21)) then
        call BEEP(1,1)
        return
    endif
endif
if (ELEPTR(1,3) .gt. 0) then
    if ((ELMNTC(ELEPTR(1,3),1).ge.char(48)) .and. (RUNG.ne.1)) then
        call BEEP(1,1)
        return
    endif
    elseif (ELMNTC(ELEPTR(1,3),1) .ge. char(45)) then
        continue
    elseif ((ELMNTC(ELEPTR(1,3),1) .ge. char(41)).and.(COLUMN .eq. 3)) then
        call BEEP(1,1)
        return
    endif
endif
if (COLUMN .eq. 12) then
    call BEEP(1,1)
    return
elseif (ELFULL) then
    call BEEP(1,1)
    return
elseif (ELEPTR(RUNG,COLUMN) .gt. 0) then
    if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(21)) then
        call BEEP(1,1)
        return
    endif
endif

end

SUBROUTINE CONTCT(MODE)
C place a NO or NC contact at the current cursor position
C (the variable MODE distinguishes between NO and NC)
C
C MODE: NO MODE=67 (F9) TYPE: X Q2=63 (F5) or Q2=64 (F6)
C   NC MODE=65 (F7) TYPE: Y Q2=65 (F7) or Q2=66 (F8)
C   C Q2=67 (F9) or Q2=68 (F10)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 TYPE
integer*2 MODE, Q1, Q2

if (COLUMN .eq. 12) then
    call BEEP(1,1)
    return
elseif (ELFULL) then
    call BEEP(1,1)
    return
endif
elseif (ELEPTR(RUNG,COLUMN) .gt. 0) then
    if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(21)) then
        call BEEP(1,1)
        return
    endif
endif
if (ELEPTR(1,3) .gt. 0) then
    if ((ELMNTC(ELEPTR(1,3),1).ge.char(48)) .and. (RUNG.ne.1)) then
        call BEEP(1,1)
        return
    endif
    elseif (ELMNTC(ELEPTR(1,3),1) .ge. char(45)) then
        continue
    elseif ((ELMNTC(ELEPTR(1,3),1) .ge. char(41)).and.(COLUMN .eq. 3)) then
        call BEEP(1,1)
        return
    endif
endif

end
elseif ((ELMNTC(ELEPTR(1,3),1).ge.char(41)) .and. (RUNG.eq.3))
   then
      call BEEP(1,1)
      return
   endif
endif

call PROMPT(6)
if (MODE .eq. 67) then
   call PRINT(PAGE,23,3,'-|',ATRIB6,5)
else
   call PRINT(PAGE,23,3,'-|',ATRIB6,5)
endif

100 call INKEY(Q1,Q2)
if (Q1 .eq. 0) then
   if ((Q2 .eq. 63) .or. (Q2 .eq. 64)) then
      (X contact)
      if (MODE .eq. 67) then
         TYPE=char(3)
      else
         TYPE=char(6)
      endif
      elseif ((Q2 .eq. 65) .or. (Q2 .eq. 66)) then
      (Y contact)
      if (MODE .eq. 67) then
         TYPE=char(4)
      else
         TYPE=char(7)
      endif
      elseif ((Q2 .eq. 67) .or. (Q2 .eq. 68)) then
      (C contact)
      if (MODE .eq. 67) then
         TYPE=char(5)
      else
         TYPE=char(8)
      endif
      elseif
      call BEEP(1,1)
      goto 100
   endif
else
   (esc)
   if (Q1 .eq. 27) then
      call PROMPT(KEY)
      return
   else
      call BEEP(2,6)
      goto 100
   endif
endif
call ADDELEITYPE)
call NRELEM(ATRIB4)
call EDIT
call MOVRGT
return
end

SUBROUTINE COIL(MODE)

place a ND or NE coil at the current cursor position
(the variable MODE distinguishes between ND and NE)

C MODE: ND MODE=68 (F10) TYPE: Y Q2=66 (F8) or Q2=65 (F7)
C NE MODE=66 (F8) Q2=68 (F10) or Q2=67 (F9)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.DLK'

character=1 TYPE
SUBROUTINE PROMPT(KEY)

    C displays the following function key assignments at bottom of screen:

    C--------------
    C PROMPT 1:  
    C blanks function display line
    C--------------
    C PROMPT 2:  

Appendix F. SYSTEM PROGRAM SOURCE LISTING
integer*2 KEY

call ERASLN(PAGE,23,'ATRIB6')
call ERASLN(PAGE,24,'ATRIB6')
1 if (KEY .eq. 1) return
goto (1,2,3,4,5,6,7,8,9,10,11,12) KEY
call BEEP(24,1)
return
2 call PRINT(PAGE,23,2,'F1 = MENU',ATRIB6,9)
call PRINT(PAGE,24,2,'F2 = EDIT',ATRIB6,9)
call PRINT(PAGE,23,17,'F3 = small box',ATRIB6,14)
call PRINT(PAGE,24,17,'F4 = large box',ATRIB6,14)
call PRINT(PAGE,23,35,'F5 = |',ATRIB6,7)
call PRINT(PAGE,24,35,'F6 = ——',ATRIB6,8)
call PRINT(PAGE,23,49,'F7 = 1/F-',ATRIB6,10)
call PRINT(PAGE,24,49,'F8 = —(/)—',ATRIB6,10)
call PRINT(PAGE,23,65,'F9 = -|',ATRIB6,11)
call PRINT(PAGE,24,65,'F10 = -( )—',ATRIB6,11)
return
3 call PRINT(PAGE,24,17,'F4 = drum',ATRIB6,9)
return
4 call PRINT(PAGE,23,17,'F3 = counter',ATRIB6,12)
call PRINT(PAGE,24,17,'F4 = timer',ATRIB6,10)
return
5 call PRINT(PAGE,23,17,'F3 = counter',ATRIB6,12)
call PRINT(PAGE,24,17,'F4 = up/down counter',ATRIB6,20)
return

Appendix F. SYSTEM PROGRAM SOURCE LISTING
SUBROUTINE SHOWNW

C displays a network (resets all screen cell pointers)
C {NOTE: set NTWPTR before calling this subroutine)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 C(5)
integer*2 BCKPTR, ELENT, I, J

call DNSCRL(0,1,22,1,4,ATRIB1)
call DNSCRL(0,1,22,6,78,ATRIB1)
do 10 J=1,22
    call LOCATE(PAGE,J,5)
call WRCHAT(PAGE,'|',ATRIB1,1)
call LOCATE(PAGE,J,79)
call WRCHAT(PAGE,'|',ATRIB1,1)
10 continue
call BINASC(NTWNUM,C(1),C(2),C(3),C(4),C(5))
call LOCATE(PAGE,25,10)
do 20 J=2,5
    if (C(J) .eq. char(0)) C(J)='0'
call WRCHAT(PAGE,C(J),ATRIB5,1)
20 continue
if (NTWPTR .eq. NNTAIL) then
    ENDLDR=.true.
call PRINT(PAGE,25,69,'lladder end ',ATRIB7,12)
endif
C {initialize all cell reference pointers & ROWCNT() to zero)
do 30 I=1,7
    ROWCNT(I)=0
do 40 J=1,12
    ELEPTR(I,J)=0
30 continue
40 continue
C {assign element pointers to appropriate screen cells)
ELENT=NETWRK(NTWPTR,1)
100 RUNG=ichar(ELMNTE(ELMNT,2))
COLUMN=ichar(ELMNTE(ELMNT,3))
if (ELMNTE(ELMNT,1) .ne. char(2)) ROWCNT(RUNG)=ROWCNT(RUNG)+1
if (ELMNTE(ELMNT,1) .lt. char(21)) then
ELEPTR(RUNG,COLUMN)=ELEMNT
elseif (ELMNTC(ELEMNT,1) .lt. char(48)) then
do 50 I=RUNG,RUNG+2
    do 60 J=COLUMN,COLUMN+2
        ELEPTR(I,J)=ELEMNT
    60 continue
50 continue
else
do 70 I=1,7
    do 80 J=3,11
        ELEPTR(I,J)=ELEMNT
    80 continue
70 continue
endif
call NRELEMIATRIBI)
call NRDATA
C (check that element is not last element of network)
if (ELEMNT .ne. NETNRK(NTNPTR,2)) then
    ELEMNT=ELMNTI(ELEMNT>2)
go to 100
endif
C (set a back reference pointer in all remaining screen cells)
if (NTNPTR .eq. NNHEAD) then
    C (network is first network in ladder)
    BCKPTR=0
else
    BCKPTR=-NETNRK(NETNRK(NTNPTR»4)»2)
endif
do 90 I=1,7
    do 15 J=1,11
        if (ELEPTR(I,J) .gt. 0) then
            if ((ELMNTC(ELEPTR(I,J),1) .ge. char(21)) .and.
                I .ne. ichar(ELMNTC(ELEPTR(I,J),2))) goto 15
            BCKPTR=ELEPTR(I,J)
        else
            ELEPTR(I,J)=BCKPTR
        endif
    15 continue
90 continue
do 25 I=1,7
    if (ELEPTR(I,12) .gt. 0) then
        BCKPTR=-ELEPTR(I,12)
    else
        ELEPTR(I,12)=BCKPTR
    endif
25 continue
RUNG=1
COLUMN=1
call HILITE(ATRIB4)
call COORD
return
end

C SUBROUTINE DELNTN
C (delete the current network)
$INCLUDE:'GLADDER.BLK'
integer*2 CURRNT, ELENT, LAST, NEXT
if (NNHEAD .eq. NNTAIL) then
    C (attempt to last network in ladder)
    call BEEP(2,6)
    return
C WARNING: To delete all networks in a ladder, use delete ladder function
endif
C (remove elements in network from element data list)
ELEMNT=NETNRK(NTNPTR,1)
RUNG=ichar(ELMNTC(ELEMNT,2))
COLUMN=ichar(ELMNTC(ELEMNT,3))
NEXT=ELMNT(I(ELEMNT,2))
LAST=NETNRK(NTNPTR,2)
100 call RMVELE
   if (ELEMNT .eq. LAST) goto 200
   ELEMNT=NEXT
   RUNG=ichar(ELMNTC(ELEMNT,2))
   COLUMN=ichar(ELMNTC(ELEMNT,3))
   NEXT=ELMNT(I(ELEMNT,2))
goto 100
C (check if deleted network was first network in ladder)
200 if (NETNRK(NTNPTR,4) .eq. 0) then
   NHHEAD=NETNRK(NTNPTR,5)
   NETNRK(NETNRK(NTNPTR,5),4)=0
   CURRENT=NETNRK(NTNPTR,5)
C (check if deleted network was last network in ladder)
   elseif (NETNRK(NTNPTR,5) .eq. -1) then
      NNTAIL=NETNRK(NTNPTR,4)
      NETNRK(NETNRK(NTNPTR,4),5)=-1
      CURRENT=NETNRK(NTNPTR,4)
   else
      NETNRK(NETNRK(NTNPTR,4),5)=NETNRK(NTNPTR,5)
      NETNRK(NETNRK(NTNPTR,5),4)=NETNRK(NTNPTR,4)
      CURRENT=NETNRK(NTNPTR,5)
   endif
C (set deleted record's bwd ptr to 0)
   NETNRK(NTNPTR,4)=0
C (set first unused record's fwd ptr to first unused record)
   NETNRK(NTNPTR,5)=NNFREE
C (set first unused record's bwd ptr to deleted record)
   NETNRK(NNTAIL,5)=CURRENT
   NETNRK(NNETAIL,4)=CURRENT
C (set first unused record ptr to deleted record)
   NNFREE=NTNPTR
   if (NNFULL) then
      if (ELEMNT .eq. ELEMAX) AND (SIZE .eq. 0)) NNFULL=.false.
   endif
   NTNPTR=CURRENT
   call SHONNN
   return
end

SUBROUTINE DELELE
   (delete an element at the current cursor position)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLLADDER.BLK'

integer*2 AND8, BCPTR, ELEMNT, ELETP

   if (ELEPTR(RUNG,COLUMN) .le. 0) then
      C (no element at current cursor position)
      call BEEP(2,6)
      return
   endif
   if (NETNRK(NTNPTR,1) .eq. NETNRK(NTNPTR,2)) then
      C (attempt to delete last element in network)
      call BEEP(2,6)
      return
   endif
C WARNING: To delete all elements in a network, use delete network function
   if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .eq. char(2)) then
      C (must use F6 to delete an OR link)
      call BEEP(2,6)
      return
   endif
   ELEMNT=ELEPTR(RUNG,COLUMN)
ELETYP=ichar(ELMNTC(ELEPTR(RUNG,COLUMN),1))
C (check if OR LINK is present on right)
if ((AND8(ELMNTC(ELEMENT,4),char(4)) .eq. 4) .and.
* (COLUMN .eq. 11)
* .or. ((COLUMN .lt. 11) .and. (ELEPTR(RUNG,COLUMN+1) .le. 0))
* .or. ((ELEPTR(RUNG,COLUMN+1) .gt. 0) .and.
* (ELMNTC(ELEPTR(RUNG,COLUMN+1),1) .eq. char(2)))
*then
ELMNTC(ELEMENT,1)=char(2)
ELMNTC(ELEMENT,4)=char(AND8(ELMNTC(ELEMENT,4),char(222)))
call HRLINK(RUNG,COLUMN,ATRIB1)
call HRLNK(RUNG,COLUMN+1,ATRIB4)
if (ELEPTR(RUNG,COLUMN+1) .gt. 0)
* ELMNTC(ELEPTR(RUNG,COLUMN+1),4)=
* char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN+1),4),char(239)))
if (COLUMN .eq. 1) then
call LOCATE(PAGE,ROM(RUNG)+1,5)
call WRCHAR(PAGE,'|',1)
else
if (ELEPTR(RUNG,COLUMN-1) .gt. 0) then
ELMNTC(ELEPTR(RUNG,COLUMN-1),4)=
* char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN-1),4),char(253))
else
call LOCATE(PAGE,ROM(RUNG)+1,COL(COLUMN)-1)
call WRCHAR(PAGE,' ',1)
endif
endif
call RNGDEC
else
call RMVLNK
BCKPTR=-ELMNT(ELEMENT,1)
call RMVELE
call SETPTR(ELEMENT,BCKPTR)
endif
C (erase element from screen)
if (ELETYP .lt. 21) then
call LOCATE(PAGE,ROM(RUNG)+COL(COLUMN))
call WRCHAR(PAGE,' ',ATRIB4,5)
call LOCATE(PAGE,ROM(RUNG)+1,ATRIB4,5)
call WRCHAR(PAGE,' ',ATRIB4,5)
elseif (ELETYP .lt. 48) then
call DNSCRRL(PAGE,ROM(RUNG)+8,ATRIB1)
call DNSCRRL(PAGE,1,22,18,72,ATRIB1)
else
call DNSCRRL(PAGE,1,22,18,72,ATRIB1)
endif
call HILITE(ATRIB4)
return
end
SUBROUTINE SETPTR(ELEMENT,BCKPTR)
C (reset reference pointers of all null element screen cells)
$INCLUDE:'QLAODER.BLK'
integer BCKPTR, ELEMENT, I, J, R
if (COLUMN .eq. 12) then
R=RUNG
goto 100
endif
doi=10 I=RUNG,7
do 20 J=1,11
if ((I .eq. RUNG) .and. (J .lt. COLUMN)) goto 20
if (ELEPTR(I,J) .eq. ELEMENT) goto 20
if (ELEPTR(I,J) .gt. 0) then
if ((ELMNT(ELEPTR(I,J),1) .ge. char(2))) .and.
* (I .ne. ichar(ELMNTC(ELEPTR(I,J),2))) goto 20
       BCKPTR=-ELEPTR(I,J)
   else
       ELEPTR(I,J)=BCKPTR
   endif
20 continue
10 continue
   R=1
100 do 30 I=R,7
      if (ELEPTR(I,12) .gt. 0) then
         BCKPTR=-ELEPTR(I,12)
      else
         ELEPTR(I,12)=BCKPTR
      endif
30 continue
   return
end

SUBROUTINE VERTCL
C (place a vertical connector at the current cursor position)

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

character*1 Q
integer*2 AND8, BCKPTR, C, DUMMY, ELEMNT, OR8

if ((COLUMN .eq. 12) .or. (RUNG .eq. 1)) then
   call BEEP(2,6)
   return
endif
if (ELEPTR(RUNG,COLUMN) .gt. 0) then
   if (ELMNTC(ELEPTR(RUNG,COLUMN),1) .ge. char(21)) then
      call BEEP(2,6)
      return
   endif
endif
if (ELEPTR(RUNG-1,COLUMN) .gt. 0) then
   if (ELMNTC(ELEPTR(RUNG-1,COLUMN),1) .ge. char(21)) then
      call BEEP(2,6)
      return
   endif
endif
if (ELEPTR(1,3) .gt. 0) then
   if (ELMNTC(ELEPTR(1,3),1) .ge. char(48)) then
      call BEEP(2,6)
      return
   elseif (ELMNTC(ELEPTR(1,3),1) .ge. char(21)) then
      if ((RUNG .lt. 5) .and. (COLUMN .lt. 6)) then
         call BEEP(2,6)
         return
      endif
   endif
endif
ELEMNT=ELEPTR(RUNG,COLUMN)
C (check if OR link is present)
call LOCATE(PAGE,ROW(RUNG),COL(COLUMN)+5)
call RDCHAT(PAGE,Q,DUMMY)
if (Q .eq. char(179)) then
   C (remove OR link)
   if (ELEMNT .gt. 0) then

if (ELMNTC(ELEMNT,1) .eq. char(2)) then
C (remove dummy OR element from element linked list)
call RMLNLK
BCKPTR=-ELMNTIlELEMNT,1)
call RMVELE
call SETPTR(ELEMNT,BCKPTR)
call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)+5)
call WRCAT(PAGE,' ',ATRIB4,1)
else
C (remove OR link from element)
ELMNTC(ELEMNT,4)=char(AND8(ELMNTC(ELEMNT,4),char(251)))
call WLNNK(L(RUNG,COLUMN),ATRIB4)
endif
if (COLUMN .lt. COLUMN) .and. (ELEPTR(RUNG,COLUMN+1).gt.0) then
C (remove OR link from element on right)
ELMNTC(ELEPTR(RUNG,COLUMN+1),4)=char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN+1),4),char(191)))
* call WLNNK(L(RUNG,COLUMN+1,ATRIB4)
else
endif
C (update link data for element above)
if (ELEPTR(RUNG-1,COLUMN).gt.0) then
ELMNTC(ELEPTR(RUNG-1,COLUMN),4)=char(AND8(ELMNTC(ELEPTR(RUNG-1,COLUMN),4),char(247)))
call WLNNK(L(RUNG-1,COLUMN),ATRIB1)
endif
C (update link data for element above and to the right)
if (COLUMN .lt. COLUMN) .and. (ELEPTR(RUNG-1,COLUMN+1).gt.0) then
ELMNTC(ELEPTR(RUNG-1,COLUMN+1),4)=char(AND8(ELMNTC(ELEPTR(RUNG-1,COLUMN+1),4),char(127)))
call WLNNK(L(RUNG-1,COLUMN+1,ATRIB1)
endif
C (add OR link)
if (ELEMENT .gt. 0) then
C (add OR link to element)
ELMNTC(ELEMNT,4)=char(OR8(ELMNTC(ELEMNT,4),char(4)))
if (COLUMN .lt. COLUMN) .and. (ELEPTR(RUNG,COLUMN+1).gt.0) then
C (add OR link to element on right)
ELMNTC(ELEPTR(RUNG,COLUMN+1),4)=char(OR8(ELMNTC(ELEPTR(RUNG,COLUMN+1),4),char(64)))
* endif
call WLNNK(L(RUNG,COLUMN),ATRIB4)
else
endif
C (add dummy OR element to element linked list)
call ADDELE(2)
call WLNNK(L(RUNG,COLUMN),ATRIB4)
endif
C (update link data for element above)
if (ELEPTR(RUNG-1,COLUMN).gt.0) then
ELMNTC(ELEPTR(RUNG-1,COLUMN),4)=
*  
char(OR8(ELMNTC(ELEPTR(RUNG-1,COLUMN),4),char(8)))  
call WRLNKR(RUNG-1,COLUMN,ATRIB1)  
endf  
C {update link data for element above and to the right}  
if ((COLUMN .lt. 11).and.(ELEPTR(RUNG-1,COLUMN+1) .gt. 0)) then  
ELMNTC(ELEPTR(RUNG-1,COLUMN+1),4)=  
  char(OR8(ELMNTC(ELEPTR(RUNG-1,COLUMN+1),4),char(128)))  
call WRLNKL(RUNG-1,COLUMN+1,ATRIB1)  
endif  
endf  
return  
end  

SUBROUTINE RMVLNK  
C { this subroutine should not be called by any }  
C { subroutine other than DELELE or VERTCL }  

$INCLUDE:'GATRIB.BLK'$INCLUDE:'QLADDER.BLK'  

integer*2 AND8, C, ELEMNT, ELETYP, I, J, OR8  
ELEMNT=ELEPTR(RUNG,COLUMN)  
ELETYP=ichar(ELMNTC(ELEMNT,1))  
C=COLUMN  
C ---———--———- contact ————-———--———-  
if (ELETYP .lt. 9) then  
if (COLUMN .eq. 1) then  
call LOCATE(PAGE,ROW(RUNG)+1,5)  
call WRCHAR(PAGE,'|',1)  
endif  
if (COLUMN .gt. 1) then  
  C {check element on left}  
  if (ELEPTR(RUNG,COLUMN-1) .gt. 0) then  
    if ((ELMNTC(ELEPTR(RUNG,COLUMN-1),1) .ge. char(21)) .and.  
      * (RUNG.ne.ichar(ELMNTC(ELEPTR(RUNG,COLUMN-1),2)))) goto 100  
    ELMNTC(ELEPTR(RUNG,COLUMN-1),4)=  
      char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN-1),4),char(253)))  
call NRLNKR(RUNG,COLUMN-1,ATRIB1)  
  else  
    if (AND8(ELMNTC(ELEMNT,4),char(64)) .eq. 64) then  
      COLUMN=COLUMN-1  
call ADDELE(2)  
      ELMNTC(ELEPTR(RUNG,COLUMN),4)=  
        char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN),4),char(253)))  
call WRLNKR(RUNG,COLUMN,ATRIB1)  
      COLUMN=C  
    else  
      call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)-1)  
call WRCHAR(PAGE,' ',1)  
    endif  
  endif  
100  if (COLUMN .lt. 11) then  
  C {check element on right}  
  if (ELEPTR(RUNG,COLUMN+1) .gt. 0) then  
    if ((ELMNTC(ELEPTR(RUNG,COLUMN+1),1) .lt. char(21)) .or.  
      * (RUNG.eq.ichar(ELMNTC(ELEPTR(RUNG,COLUMN+1),2)))) then  
      ELMNTC(ELEPTR(RUNG,COLUMN+1),4)=  
        char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN+1),4),char(239)))  
call WRLNKL(RUNG,COLUMN+1,ATRIB4)  
    else  
      call LOCATE(PAGE,ROW(RUNG)+1,COL(COLUMN)+5)  
call WRCHAT(PAGE,' ',ATRIB4,1)  
    endif  
  endif  
C ————-———-———-—-— coil ————-———-———-———
elseif (ELETYP .lt. 21) then
  call LOCATE(PAGE, ROW(RUNG)+1, 73)
  call WRCHAR(PAGE, ',', 1)
  call LOCATE(PAGE, ROW(RUNG)+1, 79)
  call WRCHAR(PAGE, '|', ATRIB4, 1)
  if (RUNG .eq. 1) then
    if (AND8(ELMNTC(ELEMNT, 4), char(128)) .eq. 128) then
      call LOCATE(PAGE, 3, 73)
      call WRCHAR(PAGE, '1', 1)
    endif
  else
    if (AND8(ELMNTC(ELEMNT, 4), char(128)) .eq. 128) then
      call LOCATE(PAGE, RON(RUNG)+1, 73)
      call WRCHAR(PAGE, '|', 1)
    else
      J = RUNG
      call LOCATE(PAGE, RON(J)+1, 73)
      call WRCHAR(PAGE, ',', 1)
      call LOCATE(PAGE, RON(J)+1, 73)
      call WRCHAR(PAGE, '|', 1)
      J = J - 1
      if (J .ge. 1) then
        if (ELEPTR(J, 12) .gt. 0) then
          ELMNTC(ELEPTR(J, 12), 4) = 
          char(AND8(ELMNTC(ELEPTR(J, 12), 4), char(128)))
          call LOCATE(PAGE, RON(J)+1, 73)
          if (J .eq. 1) then
            call WRCHAR(PAGE, '|', 1)
          else
            call WRCHAR(PAGE, '-', 1)
          endif
        else
          goto 200
        endif
      endif
    endif
  endif
endif
endif

C - - - - - - - - - - - - - - small box - - - - - - - - - - -
elseif (ELETYP .lt. 48) then
  if (ELETYP .lt. 41) then
    C (1-input box)
    I = 0
  elseif (ELETYP .lt. 45) then
    C (2-input box)
    I = 1
  else
    C (3-input box)
    I = 2
  endif
do 30 J = RUNG, RUNG+I
  if (ELEPTR(J, COLUMN-1) .gt. 0) then
    ELMNTC(ELEPTR(J, COLUMN-1), 4) = 
    char(AND8(ELMNTC(ELEPTR(J, COLUMN-1), 4), char(253)))
  endif
  30 continue
C (check element on right)
  if (ELEPTR(RUNG, COLUMN+1) .gt. 0) then
    ELMNTC(ELEPTR(RUNG, COLUMN+1), 4) = 
    char(AND8(ELMNTC(ELEPTR(RUNG, COLUMN+1), 4), char(223)))
    call NLINKL(RUNG, COLUMN+1, ATRIB1)
  endif
C - - - - - - - - - - - - - - large box - - - - - - - - - - -
else
  C (check element on left)
  if (ELEPTR(1, 2) .gt. 0) then
    ELMNTC(ELEPTR(1, 2), 4) = 
    char(AND8(ELMNTC(ELEPTR(1, 2), 4), char(253)))
  endif

Appendix F. SYSTEM PROGRAM SOURCE LISTING
SUBROUTINE WRNLNL(R, C, ATTRIB)
C {write link on screen}
C R - rung on which to write
C C - column on which to write
$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QLADDER.BLK'

INTEGER*2 AND8, ATTRIB, C, J, LFTLNK, R

LFTLNK=AND8(ELMNTCIELEPTR(R, C), 4), char(240))
if (C .lt. 12) then
  if (LFTLNK .gt. 0) then
    call LOCATE(PAGE, ROM(R)+1, COL(C)-1)
    if ((LFTLNK .eq. 16) .or. (LFTLNK .eq. 32) .or.
        (LFTLNK .eq. 48) ) then
      call NRCHAT(PAGE, '/', ATTRIB, 1)
    elseif ((LFTLNK .eq. 64) .or. (LFTLNK .eq. 128)) then
      call NRCHAT(PAGE, ' ', ATTRIB, 1)
    elseif (LFTLNK .eq. 80) then
      call NRCHAT(PAGE, 'J', ATTRIB, 1)
    elseif (LFTLNK .eq. 96) then
      call NRCHAT(PAGE, 'L', ATTRIB, 1)
    elseif (LFTLNK .eq. 112) then
      call NRCHAT(PAGE, 'I', ATTRIB, 1)
    elseif (LFTLNK .eq. 144) then
      call NRCHAT(PAGE, 'T', ATTRIB, 1)
    elseif (LFTLNK .eq. 192) then
      call NRCHAT(PAGE, '|', ATTRIB, 1)
    elseif (LFTLNK .eq. 224) then
      call NRCHAT(PAGE, '{', ATTRIB, 1)
    elseif (LFTLNK .eq. 240) then
      call NRCHAT(PAGE, '+', ATTRIB, 1)
    endif
  endif
endif
endif

C ---——— coil ·---————————— coil ·———————————— coil ·———————————— coil ·———————————— coil
else
C (COLUMN=12)
  if (LFTLNK .eq. 48) then
    call LOCATE(PAGE, ROM(R)+1, 72)
    call HRCHAR(PAGE, '-', 2)
  elseif (LFTLNK .eq. 96) then
    call LOCATE(PAGE, ROM(R)+1, 73)
    call HRCHAR(PAGE, 'L', 1)
  elseif (LFTLNK .eq. 176) then
    call LOCATE(PAGE, ROM(R)+1, 74)
    call HRCHAR(PAGE, 'T', 1)
  elseif (LFTLNK .eq. 224) then
    call LOCATE(PAGE, ROM(R)+1, 75)
    call HRCHAR(PAGE, '|', 1)
  endif
endif

Appendix F. SYSTEM PROGRAM SOURCE LISTING 195
call LOCATE(PAGE,ROW(R)+1,73)
call HRCHAR(PAGE,'|',1)
else
endif
if (((LFTLNK .eq. 96) .or. (LFTLNK .eq. 224)) then
J=R
100
call LOCATE(PAGE,ROW(J),73)
call HRCHAR(PAGE,'|',1)
call LOCATE(PAGE,ROW(J)-1,73)
call HRCHAR(PAGE,'|',1)
J=J-1
if (ELEPTR(J,12) .le. 0) then
if (J .eq. 1) then
    call LOCATE(PAGE,ROW(J)+1,73)
call HRCHAR(PAGE,'γ',1)
else
    call LOCATE(PAGE,ROW(J)+1,73)
call HRCHAR(PAGE,'|',1)
goto 100
endif
else
    call LOCATE(PAGE,ROW(J)+1,73)
if (J .eq. 1) then
    call HRCHAR(PAGE,'|',1)
else
    call HRCHAR(PAGE,'γ',1)
endif
endif
endif
endif
endif
endif
return
end

SUBROUTINE HRLNKR(R,C,ATTRIB)
C (write link on screen)
C R — rung on which to write
C C — column on which to write
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'GLADDER.BLK'

integer*2 AND8, ATTRIB, C, RGTLNK, R
RGTLNK=AND8(EMNTC(ELEPTR(R,C),4),char(15))
if (RGTLNK .gt. 0) then
    call LOCATE(PAGE,ROW(R)+1,COL(C)+5)
if ((RGTLNK .eq. 1) .or. (RGTLNK .eq. 3)) then
    call HRCHAR(PAGE,'-',ATTRIB,1)
elseif (RGTLNK .eq. 4) then
    call HRCHAR(PAGE,' ',ATTRIB,1)
elseif (RGTLNK .eq. 5) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
elseif (RGTLNK .eq. 7) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
elseif (RGTLNK .eq. 9) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
elseif (RGTLNK .eq. 11) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
elseif (RGTLNK .eq. 12) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
elseif (RGTLNK .eq. 13) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
elseif (RGTLNK .eq. 15) then
    call HRCHAR(PAGE,'\',ATTRIB,1)
endif
if ((RGTLNK .eq. 4).or.(RGTLNK .eq. 5).or. (RGTLNK .eq. 7)
or. (RGTLNK .eq. 12).or. (RGTLNK .eq. 13).or.
SUBROUTINE ADDLNK

C determine link data and add to element list

$INCLUDE:'GATRIB.BLK'
$INCLUDE:'GLADDER.BLK'

character*1 TEMP
integer*2 AND8, C, BCKPTR, ELEMNT, ELETYP, I, J, OR8, R

ELEMNT=ELEPTR(RUNG,COLUMN)
ELETYP=ichar(ELMNTC(ELEMNT,1))
R=RUNG
C=COLUMN

C ———— contact ————

if (ELETYP .lt. 9) then
  if (COLUMN .gt. 1) then
    if (ELEPTR(RUNG,COLUMN-1) .gt. 0) then
      if ((ELMNTC(ELEPTR(RUNG,COLUMN-1),1) .ge. ichar(21)) .and.
          (RUNG.ne.ichar(ELMNTC(ELEPTR(RUNG,COLUMN-1),2)))) goto 300
      if (ELETYP .ne. 2) then
        ELMNTCIELEPTR(RUNG,COLUMN-1),4)=char(OR8(ELMNTC(ELEMNT,4),char(2)))
        TEMP=charl16*(AND8(ELMNTC(ELEMNT,4),char(15))))
        ELMNTClELEMNT,4)=char(OR8(ELMNTC(ELEMNT,4),TEMP))
      endif
      elseif (RUNG .lt. 7) then
        if (ELEPTR(RUNG+1,COLUMN-1) .gt. 0) then
          if (AND8(ELMNTC(ELEPTR(RUNG+1,COLUMN-1),4),char(128)))
            ELMNTC(ELEMNT,4)=char(OR8(ELMNTC(ELEMNT,4),char(128)))
          endif
        endif
      endif
  endif
endif

if (COLUMN .lt. 11) then
  if (COLUMN .lt. 11) then
    if (ELEPTR(RUNG+1,COLUMN+1) .gt. 0) then
      if((ELMNTC(ELEPTR(RUNG+1,COLUMN+1),1) .ge. ichar(21)) .and.
          (RUNG.ne.ichar(ELMNTC(ELEPTR(RUNG,COLUMN+1),2)))) goto 400
      if (ELETYP .ne. 2) then
        ELMNTC(ELEPTR(RUNG,COLUMN+1),4)=char(OR8(ELMNTC(ELEMNT,4),char(16)))
        TEMP=charl16*(AND8(ELMNTC(ELEMNT,4),char(16))))
      endif
      endif
  endif
endif

end

C ———— return ————

end
* char((AND8(ELMNTC(ELEPTR(RUNG, COLUMN+1), 4), char(240)))/16)
  ELMNTC(ELEMNT, 4)=char(OR8(ELMNTC(ELEMNT, 4), TEMP))
else if (RUNG .lt. 7) then
  C (check element below & to the right)
  if (ELEPTR(RUNG+1, COLUMN+1) .gt. 0) then
    if (AND8(ELMNTC(ELEPTR(RUNG+1, COLUMN+1), 4), char(64))
      * .eq. 64) ELMNTC(ELEMNT, 4) =
      * char(OR8(ELMNTC(ELEMNT, 4), char(8)))
    endif
  endif
endif
400 endif
C ------------------- coil ------------------------
elseif (ELETYP .lt. 21) then
  if (RUNG .eq. 1) then
    ELMNTC(ELEMNT, 4)=char(48)
    do J=2,7
      if (ELEPTR(J, 12) .gt. 0) then
        ELMNTC(ELEMNT, 4)=char(176)
        goto 200
      endif
    10 continue
  else
    ELMNTC(ELEMNT, 4)=char(96)
    if (RUNG .lt. 7) then
      do 20 J=RUNG+1,7
        if (ELEPTR(J, 12) .gt. 0) then
          ELMNTC(ELEMNT, 4)=char(224)
          goto 100
        endif
      20 continue
    endif
  endif
100 endif
do 50 J=RUNG-1,1,-1
  if (ELEPTR(J, 12) .gt. 0) then
    ELMNTC(ELEPTR(J, 12), 4)=
    * char(OR8(ELMNTC(ELEPTR(J, 12), 4), char(128)))
    goto 200
  endif
50 continue
200 endif
C ------------------- small box ---------------------
elseif (ELETYP .lt. 48) then
  ELMNTC(ELEMNT, 4)=char(51)
  elseif (ELETYP .lt. 41) then
    I=0
  elseif (ELETYP .lt. 45) then
    I=1
  else
    I=2
  endif
C (check elements on left)
  do 50 RUNG=R+I
    if (ELEPTR(RUNG, C-1) .gt. 0) ELMNTC(ELEPTR(RUNG, C-1), 4)=
    * char(OR8(ELMNTC(ELEPTR(RUNG, C-1), 4), char(2)))
  if (I .gt. 0) then
    do 40 COLUMN=1,2
      if (ELEPTR(RUNG, COLUMN) .gt. 0) then
        if (ELMNTC(ELEPTR(RUNG, COLUMN), 1) .eq. char(2)) then
          BCKPTR=-ELMNTI(ELEPTR(RUNG, COLUMN), 1)
          call RMVELE
          ELEPTR(RUNG, COLUMN)=BCKPTR
          call SETPTR(ELEPTR(RUNG, COLUMN), BCKPTR)
        endif
      endif
    40 continue
  endif
50 continue
Appendix F. SYSTEM PROGRAM SOURCE LISTING 198
elseif (AND8(ELMNTC(ELEPTR(RUNG,COLUMN),4),char(204)),ge.4)
    * ELMNTC(ELEPTR(RUNG,COLUMN),4)=char(AND8(ELMNTC(ELEPTR(RUNG,COLUMN),4),char(51)))
call WRLNK(L(RUNG,COLUMN),ATRIB1)
call WRLNKR(RUNG,COLUMN,ATRIB1)
endif
if (RUNG.gt.1) then
    call LOCATE(PAGE,ROW(RUNG)-1,COL(COLUMN)+5)
call WRCHAR(PAGE,' ',1)
call LOCATE(PAGE,ROW(RUNG),COL(COLUMN)+5)
call WRCHAR(PAGE,' ',1)
endif
endif
40 continue
endif
50 continue
do 60 RUNG=R+I+1,R+2
if (ELEPTR(RUNG,C-1).gt.0) then
    * ELMNTC(ELEPTR(RUNG,C-1),4)=char(AND8(ELMNTC(ELEPTR(RUNG,C-1),4),char(253)))
    COLUMN=C-1
call WRLNKR(RUNG,C-1,ATRIB1)
endif
60 continue
if (I.gt.0) then
    do 70 COLUMN=1,2
if (ELEPTR(R+I+1,COLUMN).gt.0) then
    if (ELMNTC(ELEPTR(R+I+1,COLUMN),1).eq. char(2)) then
        BCKPTR=—ELMNT1(ELEPTR(R+I+1,COLUMN),1)
        RUNG=R+I+1
        call RMVELE
        ELEPTR(R+I+1,COLUMN)=BCKPTR
        call SETPTR(ELEPTR(R+I+1,COLUMN),BCKPTR)
    elseif (ANO8(ELMNTC(ELEPTR(R+I+1,COLUMN),4),char(68)).ge.4)
    * ELMNTC(ELEPTR(R+I+1,COLUMN),4)=char(ANO8(ELMNTC(ELEPTR(R+I+1,COLUMN),4),char(187)))
call NRLNKL(R+I+1,COLUMN,ATRIB1)
call NRLNKR(R+I+1,COLUMN,ATRIB1)
endif
    call LOCATE(PAGE,RON(R+I+1)-1,COL(COLUMN)+5)
call WRCHAR(PAGE,' ',1)
call LOCATE(PAGE,RON(R+I+1),COL(COLUMN)+5)
call WRCHAR(PAGE,' ',1)
endif
70 continue
endif
C (check element below)
RUNG=R+3
if (ELEPTR(R+3,C-1).gt.0) then
    if (ELMNTC(ELEPTR(R+3,C-1),1).eq. char(2)) then
        COLUMN=C-1
        BCKPTR=—ELMNT1(ELEPTR(R+3,C-1),1)
        call RMVELE
        ELEPTR(R+3,C-1)=BCKPTR
        call SETPTR(ELEPTR(R+3,C-1),BCKPTR)
    elseif (ANO8(ELMNTC(ELEPTR(R+3,C-1),4),char(68)).ge.4)
    * ELMNTC(ELEPTR(R+3,C-1),4)=char(ANO8(ELMNTC(ELEPTR(R+3,C-1),4),char(187)))
call NRLNKL(R+3,C-1,ATRIB1)
call NRLNKR(R+3,C-1,ATRIB1)
endif
    call LOCATE(PAGE,ROW(R+3)-1,COL(COLUMN)+5)
call WRCHAR(PAGE,' ',1)
endif
C (check element below on left for right OR link)
if (ELEPTR(R+3,C).gt.0) then

if (ELMNTC(ELEPTR(R+3,C),1) .eq. char(2)) then
  COLUMN=C
  BCKPTR=ELMNTI(ELEPTR(R+3,C),1)
call RMVELE
  ELEPTR(R+3,C)=BCKPTR
call SETPTR(ELEPTR(R+3,C),BCKPTR)
else
  if (I .eq. 2) then
    if (AND8(ELMNTC(ELEPTR(R+3,C),4),char(68)) .ge. 4)
      * ELMNTC(ELEPTR(R+3,C),4)=
      x char(AND8(ELMNTC(ELEPTR(R+3,C),4),char(187)))
call WRLNKR(R+3,C,ATRIB1)
    else
      if (AND8(ELMNTC(ELEPTR(R+3,C),4),char(4)) .eq. 4)
        * ELMNTC(ELEPTR(R+3,C),4)=
        x char(AND8(ELMNTC(ELEPTR(R+3,C),4),char(251)))
call WRLNKR(R+3,C,ATRIB1)
    endif
  endif
  call LOCATE(PAGE,ROW(R+3),COL(C)+5)
call NRCHAR(PAGE,' °»l) endif
C {check element below on right for left OR link}
if (ELEPTR(R+3,C+2) .gt. 0) then
  if (AND8(ELMNTC(ELEPTR(R+3,C+2),4),char(64)) .eq. 64)
    * ELMNTC(ELEPTR(R+3,C+2),4)=
    x char(AND8(ELMNTC(ELEPTR(R+3,C+2),4),char(191)))
call WRLNKL(R+3,C+2,ATRIB1)
  endif
  call LOCATE(PAGE,ROW(R+3),COL(C+2)-1)
call NRCHAR(PAGE,' ',1) endif
C {check elements on right}
if (ELEPTR(R,C+3) .gt. 0) then
  ELMNTC(ELEPTR(R,C+3),4)=
  * char(ORB8(ELMNTC(ELEPTR(R,C+3),4),char(16)))
  ELMNTC(ELEPTR(R,C+3),4)=
  * char(AND8(ELMNTC(ELEPTR(R,C+3),4),char(63)))
  endif
if (ELEPTR(R+1,C+3) .gt. 0) then
  ELMNTC(ELEPTR(R+1,C+3),4)=
  * char(AND8(ELMNTC(ELEPTR(R+1,C+3),4),char(47)))
call WRLNKL(R+1,C+3,ATRIB1)
endif
if (ELEPTR(R+2,C+3) .gt. 0) then
  ELMNTC(ELEPTR(R+2,C+3),4)=
  * char(AND8(ELMNTC(ELEPTR(R+2,C+3),4),char(175)))
call WRLNKL(R+2,C+3,ATRIB1)
endif
C - - - - - - - - - - large box - - - - - - - - - - -
else
  ELMNTC(ELEMNT,4)=char(51)
C {check element on left}
if (ELEPTR(1,2) .gt. 0) then
  ELMNTC(ELEPTR(1,2),4)=char(ORB8(ELMNTC(ELEPTR(1,2),4),char(2)))
  ELMNTC(ELEPTR(1,2),4)=
  * char(AND8(ELMNTC(ELEPTR(1,2),4),char(119)))
call WRLNKR(1,2,ATRIB1)
endif
if (ELEPTR(1,1) .gt. 0) then
  ELMNTC(ELEPTR(1,1),4)=
  * char(AND8(ELMNTC(ELEPTR(1,1),4),char(247)))
call WRLNKR(1,1,ATRIB1)
endif
endif
RUNG=R
COLUMN=C
return
SUBROUTINE ADDELE(ELETYP)
C add an element to the element data list

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

character*1 ELETYP
integer*2 BCKPTR, ELEMNT, I, J, OR8

if ((ELETYP.ge.21).and.(ELEPTR(RUNG,COLUMN).gt.0)) then
  BCKPTR=-ELMNT(ELEPTR(RUNG,COLUMN),1)
call RMVELE
ELEPTR(RUNG,COLUMN)=BCKPTR
endif

if (ELEPTR(RUNG,COLUMN).eq.0) then
  C {replace old element}
  ELEMNT=ELEPTR(RUNG,COLUMN)
  if (ELEMNT.EQ.ELEMNT,1).eq.char(2) then
    if (COLUMN .eq. 1) then
      ELEMNT(ELEMNT,1)=char(OR8(ELEMNT(ELEMNT,4),char(1)))
    else
      ELEMNT(ELEMNT,1)=char(OR8(ELEMNT(ELEMNT,4),char(33)))
    endif
  endif
  C {remove type 2 element on left}
  ELMNTC(ELEMNT,4)=char(OR8(ELEMNTC(ELEMNT,4),char(64)))
  J=COLUMN
  COLUMN=COLUMN-1
  BCKPTR=-ELMNTI(ELEPTR(RUNG,COLUMN),1)
call RMVELE
ELEPTR(RUNG,COLUMN)=BCKPTR
  COLUMN=J
endif
endif
ELMNTC(ELEMNT,1)=ELETYP
if (ELMNTI(ELEMNT,3).gt.0) then
  ELMNTI(ELEMNT,3)=0
  if (ELMNTI(ELEMNT,1).eq.char(5)).and.
    *(ELMNTI(ELEMNT,1).eq.char(8)).and.
    *(ELMNTI(ELEMNT,1).eq.char(11)).and.
    *(ELMNTI(ELEMNT,1).eq.char(14))) then
    TYPREF(ELMNTI(ELEMNT,3))=' '
  endif
endif
else
  (add new element)
  ELEMNT=ELFREE
C (set unused record pointer to next unused record)
  ELFREE=ELMNTI(ELFREE,2)
C (check if linked list is full)
if (ELFREE .eq. -1) then
  ELMFULL=.true.
call PRINT(PAGE,25,56,'ladder full ',ATRIB7,13)
else
  (set bkwd ptr of first unused record to 0)
  ELMNTI(ELFREE,1)=0
endif
ELMNTI(ELEMNT,1)=ELEPTR(RUNG,COLUMN)
C (set other forward & backward pointers)
if (ELEPTR(RUNG,COLUMN) .eq. 0) then

C {new element is first element in ladder}
C {set new record’s fwd ptr to previous first element}
  ELMNTI(ELEMNT,2)=ELHEAD
C {set first ladder element ptr to new element}
  ELHEAD=ELEMNT
C {set following record’s bkwrd ptr to new record}
  ELMNTI(ELMNTI(ELEMNT,1),2)=ELEMNT
elseif (ELMNTI(ELMNTI(ELEMNT,1),2) .eq. -1) then

C {new element is last element in ladder}
C {set new record’s fwd ptr to indicate it is last record}
  ELMNTI(ELEMNT,2)=-1
C {set preceding record’s fwd ptr to new record}
  ELMNTI(ELMNTI(ELEMNT,1),2)=ELEMNT
else

C {set new record’s fwd ptr to preceding record’s fwd ptr}
  ELMNTI(ELEMNT,2)=ELMNTI(ELMNTI(ELEMNT,1),2)
C {set preceding record’s fwd ptr to new record}
  ELMNTI(ELMNTI(ELEMNT,1),2)=ELEMNT
C {set following record’s bkwrd ptr to new record}
  ELMNTI(ELMNTI(ELEMNT,2),1)=ELEMNT
endif
C {check if following element was first element in network}
if (ELMNTI(ELEMNT,2) .eq. NETNRK(NTNPTR,1)) then
  C {set first element in network ptr to new element}
  NETNRK(NTNPTR>1)=ELEMNT
endif
C {check if preceding element was last element in network}
if (ELMNTI(ELEMNT,1) .eq. NETNRK(NTNPTR,2)) then
  C {set last element in network ptr to new element}
  NETNRK(NTNPTR,2)=ELEMNT
endif
C {set screen call ptr to new record}
if (ELETYP .lt. char(21)) then
  ELEPTR(RUNG,COLUMN)=ELEMNT
else
  if (ELETYP .lt. char(48)) then
    SBXTOT=SBXTOT+1
  else
    LBXTOT=LBXTOT+1
  endif
  call BOXPTR(ELEMNT,ELETYP)
endif
BCKPTR=-ELEMNT
call SETPTR(ELEMNT,BCKPTR)
C {don’t increment row counter or add link if adding a TYPE 2 element}
if (ELETYP .ne. char(2)) then
  RONCNT(RUNG)=RONCNT(RUNG)+1
  if (RONCNT(RUNG) .eq. 1) NETNRK(NTNPTR,3)=NETNRK(NTNPTR,3)+1
endif
C {add new record data}
ELMNTC(ELEMNT,1)=ELETYP
ELMNTC(ELEMNT,2)=char(RUNG)
ELMNTC(ELEMNT,3)=char(COLUMN)
if (ELETYP .eq. char(2)) then
  ELMNTC(ELEMNT,4)=char(4)
elseif (ELETYP .lt. char(21)) then
  if (COLUMN .eq. 1) then
    ELMNTC(ELEMNT,4)=char(1)
  else
    ELMNTC(ELEMNT,4)=char(33)
  endif
endif
endif
call ADDLNK
return
end
SUBROUTINE BOXPT(RELEMNT,ELEMENT,ELETYP)
C (This is a subroutine of SUBROUTINE ADDELE. It sets the cell )
C (reference pointers for all cells covered by a box type element)
C (and deletes any elements covered by the box type element. It )
C (also assigns a pointer to the box data record in SBOX or LBOX)

$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

character*1 ELEMENT
integer*2 C, ELEMENT, J, R

R=RUNG
C=COLUMN

if (ELETYP .ge. char(48)) then
  (assign LBOX pointer)
  J=1
  100 if (LBOX(J,1) .eq. 0) then
      ELMNTI(ELEMENT,3)=J
  else
      J=J+1
      if (J .gt. LBXMAX) then
          call BEEP(1,1)
          call LOCATE(2,18,10)
          write(*,*), 'ERROR in BOXPT: J > LBXMAX'
      else
          goto 100
      endif
  endif

  (set cell reference pointers & delete elements)
  do 20 RUNG=1,7
    if (RUNG .gt. 1) then
      (remove any elements below rung 1 in columns 1 or 2)
      if (ELEPTR(RUNG,1) .gt. 0) then
          COLUMN=1
          call RMVELE
      endif
      if (ELEPTR(RUNG,2) .gt. 0) then
          COLUMN=2
          call RMVELE
      endif
    endif
    do 30 COLUMN=C,C+2
      if (ELEPTR(RUNG,COLUMN) .gt. 0) .and. 
      (ELEPTR(RUNG,COLUMN) .ne. ELEMENT) call RMVELE
      ELEPTR(RUNG,COLUMN)=ELEMENT
      30 continue
  20 continue
else
  (assign SBOX pointer)
  J=1
  200 if (S8OX(J,1) .eq. 0) then
      ELMNTI(ELEMENT,3)=J
  else
      J=J+1
      if (J .gt. SBXMAX) then
          call BEEP(1,1)
          call LOCATE(2,18,10)
          write(*,*), 'ERROR in BOXPT: J > SBXMAX'
      else
          goto 200
      endif
  endif

  (set cell reference pointers & delete elements)
  do 40 RUNG=R,R+2
    do 30 COLUMN=C,C+2
      if (ELEPTR(RUNG,COLUMN) .gt. 0) .and. 
      (ELEPTR(RUNG,COLUMN) .ne. ELEMENT) call RMVELE
  30 continue
  40 continue
end subroutine

Appendix F. SYSTEM PROGRAM SOURCE LISTING
ELEPPRUNG,COLUMN)=ELEMNT
30 continue
40 continue
eif
RUNG=R
COLUMN=C
return
end

SUBROUTINE KYBDIN(NUMBER,Q,FIELD,H,FLDROW,FLDCOL,KEYCOD)
C KEYBOARD INPUT SUBROUTINE
C
C TABLE OF RETURN CODES
C
<table>
<thead>
<tr>
<th>KEYCOD</th>
<th>KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CURSOR UP</td>
</tr>
<tr>
<td>2</td>
<td>CURSOR DOWN</td>
</tr>
<tr>
<td>3</td>
<td>CURSOR LEFT</td>
</tr>
<tr>
<td>4</td>
<td>CURSOR RIGHT</td>
</tr>
<tr>
<td>5</td>
<td>SHIFT TAB</td>
</tr>
<tr>
<td>6</td>
<td>TAB</td>
</tr>
<tr>
<td>7</td>
<td>CARRIAGE RETURN</td>
</tr>
<tr>
<td>8</td>
<td>ESC</td>
</tr>
<tr>
<td>9</td>
<td>F2</td>
</tr>
<tr>
<td>10</td>
<td>SPACE BAR</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td>Y</td>
</tr>
<tr>
<td>13</td>
<td>C</td>
</tr>
</tbody>
</table>

C
TEXT INPUT FIELD DIAGRAM

C

C VARIABLES:
C FIELD - length of input field
C H - text input: number of characters entered
numeric input: number of field positions remaining
C KEYCOD - return code that indicates which key was pressed
C FLDRON - row in which input field appears
C FLDCOL - column in which input field begins
C Q1 - ASCII code (or 0 for special keys) of the key that was pressed
C Q2 - scan code (or ASCII code for special keys) of the key that was pressed
C Qt - text array of the sequence of input characters
C NUMBER - numeric input indicator

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'

logical*2 NUMBER
character*1 Q(FIELD)
integer*2 FIELD, FLDCOL, FLDROW, H, I, KEYCOD, Q1, Q2

call LOCATE(PAGE,FLDRON,FLDCOL+FIELD-1)
if (NUMBER) call CSRON
100 call INKEY(Q1,Q2)
if (Q1 .eq. 0) then
C ----—-—-—- - - - extended code keys - - - - - - - -
if (Q2 .eq. 15) then
C (shift tab)
KEYCOD=5
elseif (Q2 .eq. 60) then
C (function key F2)
KEYCOD=9
else if (Q2 .eq. 72) then
  (cursor up)
  KEYCOD=1
C
else if (Q2 .eq. 80) then
  (cursor down)
  KEYCOD=2
C
else if (Q2 .eq. 75) then
  (cursor left)
  KEYCOD=3
C
else if (Q2 .eq. 77) then
  (cursor right)
  KEYCOD=4
else
  call BEEP(FLDROW,FLDCOL+FIELD-1)
  if (NUMBER) call CSRON
  goto 100
endif
else
C —-------——---- ASCII keys --—-----——------
  if (Q1 .eq. 9) then
    (tab)
    KEYCOD=6
  elseif (Q1 .eq. 13) then
    (carriage return)
    KEYCOD=7
  elseif (Q1 .eq. 27) then
    (esc)
    KEYCOD=8
  elseif (Q1 .eq. 32) then
    (space bar)
    KEYCOD=10
  elseif ((Q1 .eq. 88) .or. (Q1 .eq. 120)) then
    (X)
    KEYCOD=11
  elseif ((Q1 .eq. 89) .or. (Q1 .eq. 121)) then
    (Y)
    KEYCOD=12
  elseif (Q1 .eq. 67) .or. (Q1 .eq. 99)) then
    (C)
    KEYCOD=13
else if (NUMBER) then
C ————-———- numeric input ————-———
  if ((H .eq. 5-FIELD) .and. (Q1 .ne. 8)) then
    call BEEP(FLDROW,FLDCOL+FIELD-1)
    goto 100
  endif
C
  (backspace)
  if (Q1 .eq. 8) then
    (no previous input)
    if (H .eq. 5) then
      call BEEP(FLDROW,FLDCOL+FIELD-1)
      call CSRON
      goto 100
    C
    (previous input)
  else
    H=H+1
    do 20 I=5,H+1,-1
      Q(I)=Q(I-1)
    20 continue
    Q(H)=char(0)
    call CSROFF
    call LOCATE(PAGE,FLDROW,FLDCOL+FIELD+H-6)
    call WRCHAR(PAGE,' ' ,1)
    do 30 I=H+1,5
      call WRCHAR(PAGE,Q(I),1)
    30 continue
    call LOCATE(PAGE,FLDROW,FLDCOL+FIELD-1)
Appendix F. SYSTEM PROGRAM SOURCE LISTING 205
if (H .eq. 6-FIELD) then
    (erase blinking carriage return)
call LOCATE(PAGE, FLDROM, FLDCOL+FIELD+1)
call NRCHAT(PAGE, ' ', ATRIB6, 3)
call LOCATE(PAGE, FLDROM, FLDCOL+FIELD-1)
endif
    call CSRON
    goto 100
endif

C  (check for non-numeric input)
if ((Q1 .lt. 48).or.(Q1 .gt. 57)) then
    call BEEP(FLDROM, FLDCOL+FIELD-1)
call CSRON
    goto 100
else
    do 40 I=H,4
         Q(I)=Q(I+1)
    40 continue
    Q(5)=char(Q1)
call LOCATE(PAGE, FLDROM, FLDCOL+FIELD+H-6)
    do 50 I=H,5
         call NRCHAR(PAGE, Q(I), 1)
    50 continue
    H=H-1
call LOCATE(PAGE, FLDROM, FLDCOL+FIELD-1)
endif
C  (field full)
if (H .eq. 5-FIELD) then
    call CSROFF
call PRINT(PAGE, FLDROM, FLDCOL+FIELD+1, '<-J', ATRIB0, 3)
endif
goto 100
else
    call BEEP(1,1)
goto 100
endif
endif

C  (erase blinking carriage return)
call LOCATE(PAGE, FLDROM, FLDCOL+FIELD+1)
call NRCHAT(PAGE, ' ', ATRIB6, 3)
call LOCATE(PAGE, FLDROM, FLDCOL+FIELD-1)
return

d SUBROUTINE WRDATA
C  (write element data after element has been drawn)
$INCLUDE:'QATRIB.BLK' $INCLUDE:'QLADDER.BLK' $INCLUDE:'QBOXES.BLK'
character*1 Q(5)
integer*2 AND16, BOXPTR, ELMNT, ELETYP, I, J, K
ELEMNT=ELEPTR(RUNG,COLUMN)
ELETYP=ichar(ELMNTC(ELEMNT,1))
if (ELMNTI(ELEMNT,3) .eq. 0) return
BOXPTR=ELMNTI(ELEMNT,3)
C  - - - - - - - - - - - coil or contact - - - - - - - - - - - - - - - - - - - - -
if (ELETYP .lt. 21) then
    call WRVAL(RUNG, COLUMN)+1, ELMNTI(ELEMNT, 4)
C  - - - - - - - - - - - small box - - - - - - - - - - - - - - - - - - - - - -
elseif (ELETYP .lt. 48) then
C  (write reference number)
    if (ELETYP .eq. 41) then
call NRVAL(3, COL(COLUMN)+11, SBOX(BOXPTR,1), 3)
elseif (ELETYP .eq. 42) then
    call HRVALI3, COL(COLUMN)+9, SBOX(BOXPTR) > 3
elseif (ELETYP .eq. 47) then
    call NRVAL(4, COL(COLUMN)+11, SBOX(BOXPTR,1), 3)
endif

C {write preset value}
call NRVAL(7, COL(COLUMN)+4, $BOX(BOXPTR) > 3)
C {write zero coil reference}
if ((ELETYP .eq. 47) .and. (SBOX(BOXPTR,3) .gt. 0)) then
    call LOCATE(PAGE,9, COL(COLUMN)+4)
call NRCHAR(PAGE, SBOXC(BOXPTR), 1)
call NRVAL(9, COL(COLUMN)+5, SBOX(BOXPTR,3), 3)
endif

C ------------—--— large box ------------------------
else
    C {write reference number}
call NRVAL(2,26, LBOX(BOXPTR,1), 2)
    C {write reset step}
call NRVAL(3,34, LBOX(BOXPTR,2), 2)
    C {write seconds per count}
if (LBOX(BOXPTR,3) .gt. 0) then
    call BINAASC(LBOX(BOXPTR,3), Q(1), Q(2), Q(3), Q(4), Q(5))
call LOCATE(PAGE,4,33)
do 10 J=1,5
    if (Q(J) .eq. char(0)) then
        if (J .eq. 1) goto 10
        Q(J)='0'
    endif
    call NRCHAR(PAGE, Q(J), 1)
10 continue
endif
    C {write counts per step}
do 30 I=4,19
    if (LBOX(BOXPTR,I) .gt. 0) then
        call BINAASC(LBOX(BOXPTR,I), Q(1), Q(2), Q(3), Q(4), Q(5))
call LOCATE(PAGE,I+2,30)
do 20 J=1,5
        if (Q(J) .eq. char(0)) Q(J)=''
call NRCHAR(PAGE, Q(J), 1)
20 continue
endif
    C {write controlled output coil references}
do 50 I=20,34
    if (LBOX(BOXPTR,I) .gt. 0) then
        call LOCATE(PAGE,1,39+(I-19)*2)
call BINAASC(LBOX(BOXPTR,I), K=1)
do 40 J=1,5
        if (Q(J) .ne. char(0)) then
            K=K+1
        endif
call LOCATE(PAGE, K,39+(I-19)*2)
call NRCHAR(PAGE, Q(J), 1)
40 continue
endif
    C {write output sequence bits for each controlled coil}
do 70 I=1,16
do 60 J=0,16
    call LOCATE(PAGE, I+5,2*J+1)
    if (AND16(LBOX(BOXPTR,I+34),2*J) .eq. 2*J) then
        call NRCHAR(PAGE, '! ', 1)
    endif
60 continue
70 continue
endif
SUBROUTINE WRVAL(ROW, COL, VALUE, LENGTH)

C (left justifies and writes value using current attribute)

$INCLUDE:'QATRIB.BLK'

character*1 Q(5)

integer*2 COL, J, LENGTH, ROW, VALUE

if (VALUE .gt. 0) then
   call BINASC(VALUE, Q(1), Q(2), Q(3), Q(4), Q(5))
   call LOCATE(PAGE, ROW, COL)
   do 10 J=6-LENGTH,5
      if (Q(J) .ne. char(0)) then
         call WRCHAR(PAGE, Q(J), 1)
      endif
   10 continue
   endif

return
end

SUBROUTINE EDIT

$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

character*1 Q(5), LIMIT(5)

integer*2 AND16, C, ELETYP, FLDCOL, H, J, K, KEYCOD, OR16, R, TEMP

logical*2 XYFLAG

integer*2 COIL, ELEMNT, STEP

common/EDIT/ COIL, ELEMNT, STEP

data LIMIT/ '3', '2', '7', '6', '7' /

if (ELEPTR(RUNG, COLUMN) .le. 0) then
   (no element at current cursor position)
   call BEEP(1, 1)
   return
endif

ELEMNT=ELEPTR(RUNG, COLUMN)
ELETYP=ichar(ELMNTC(ELEMNT; 1))

if ((ELETYP .eq. 1) .or. (ELETYP .eq. 2)) then
   (cannot edit horizontal or vertical elements)
   call BEEP(1, 1)
   return
endif
call PROMPT(12)

C -------- coil or contact --------

if (ELETYP .lt. 21) then
   call HIVAL3(ROW, RUNG, (COLUMN) + 1, ELMNTI(ELEMNT, 3), 4)
   call ERASLN(PAGE, 23, ATRIB6)
100 call CSROFF
   if (ELETYP .lt. 9) then
      call PRINT(PAGE, 23, 2, 'Enter the reference number for this contact:
      ' , ATRIB6, 49)
      FLDCOL=47
   else
      call PRINT(PAGE, 23, 2, 'Enter the reference number for this coil:
      ' , ATRIB6, 6)

Appendix F. SYSTEM PROGRAM SOURCE LISTING 208
IDGE=44
endif

do J=1,5
Q(J)=char(0)
10 continue
H=5
200 call KYBDIN(.true..Q(4),H,23,FLDCOL,KEYCOD)
if(KEYCOD .eq. 7) then
C
(carriage return)
call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),TEMP)
if ( (ELETYP .eq. 5) .or. (ELETYP .eq. 8) .or. (ELETYP .eq. 11)
 .or. (ELETYP .eq. 14)) then
XYFLAG=.false.
J=511
else
XYFLAG=.true.
J=1023
endif
if (TEMP .gt. J) then
C
(value exceeds limit)
call BEEP(23,FLDCOL)
goto 100
endif
if (XYFLAG) then
if (TEMP .eq. 0) then
TYPREF(ELMNT(ELEMNT,3))=' '
else
if ((ELETYP .eq. 3) .or. (ELETYP .eq. 6)) then
if (TYPREF(TEMP) .eq. 'Y') then
C
(this reference number already assigned)
call BEEP(23,FLDCOL)
goto 100
else
TYPREF(TEMP)='X'
endif
else
if (TYPREF(TEMP) .eq. 'X') then
C
(this reference number already assigned)
call BEEP(23,FLDCOL)
goto 100
else
TYPREF(TEMP)='Y'
endif
endif
endif
ELMNT(ELEMNT,3)=TEMP
call HVIAL4(ROW(RUNG),COL(COLUMN)+1,ELMNT(ELEMNT,3),4,Q)
elseif(KEYCOD .eq. 8) then
(esc)
goto 100
elseif(KEYCOD .eq. 9) then
(F2 - exit edit)
call BINASC(ELMNT(ELEMNT,3),Q(1),Q(2),Q(3),Q(4),Q(5))
call HVIAL4(ROW(RUNG),COL(COLUMN)+1,ELMNT(ELEMNT,3),4,Q)
else
C
small box
C-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
call BEEP(23,FLDCOL)
goto 200
endif
elseif (ELETYP .lt. 48) then
300 if (ELETYP .eq. 41) then
R=3
C=COL(COLUMN)+11
elseif (ELETYP .eq. 42) then
R=3
C=COL(COLUMN)+9
elseif (ELETYP .eq. 47) then
R=4
C=COL(COLUMN)+11
endif

call HIVAL3(R,C,SBOX(ELMNTI(ELEMNT,3),1),3)
call ERASLN(PAGE,23,ATRIB6)

400
call CSROFF
if ((ELETYP .eq. 41) .or. (ELETYP .eq. 47)) then
call PRINT(PAGE,23,2,'Enter the reference number for this counter: *
* ',ATRIB6,48)
FLDCOL=47
elseif (ELETYP .eq. 42) then
call PRINT(PAGE,23,2,'Enter the reference number for this timer: *
* ',ATRIB6,46)
FLDCOL=45
do J=1,5
Q(J)=char(0)
20 continue
H=5

500
call KYBDIN (.true.,Q,3,H,23,FLDCOL,KEYCOD)
C
{cursor up,cursor down,carriage return,F2)
TEMP=SBOX(ELMNTI(ELEMNT,3),1)
if (((KEYCOD .ne. 7) .and. (Q(5) .eq. char(0))) .or. *(KEYCOD .eq. 9)) then
call BINSAS(SBOX(ELMNTI(ELEMNT,3),1),Q(1),Q(2),Q(3),Q(4),Q(5))
else
call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),TEMP)
endif
if (TEMP .gt. 128) then
C
{value exceeds limit}
call BEEP(23,FLDCOL)
goto 400
elseif ((KEYCOD .eq. 7) .and. (KEYCOD .eq. 9)) then
if (SBXREF(TEMP) .gt. 0) then
C
{reference number already assigned}
call BEEP(23,FLDCOL)
goto 400
endif
SBXREF(SBOX(ELMNTI(ELEMNT,3),1))=0
SBOX(ELMNTI(ELEMNT,3),1)=TEMP
if (TEMP .gt. 0) SBXREF(TEMP)=ELEMNT
call HIVAL4(R,C,SBOX(ELMNTI(ELEMNT,3),1),3,Q)
if (KEYCOD .eq. 9) then
call PROMPT(KEY)
return
elseif ((KEYCOD .eq. 1) .and. (ELETYP .eq. 47)) then
goto 250
else
goto 700
endif
elseif (KEYCOD .eq. 8) then
(C
{esc}
goto 400
else
call BEEP(23,FLDCOL)
goto 500
endif
C
-enter preset value

700
call HIVALS(7,COL(COLUMN)+4,SBOX(ELMNTI(ELEMNT,3),2),5)
call ERASLN(PAGE,23,ATRIB6)

800
call CSROFF
if ((ELETYP .eq. 41) .or. (ELETYP .eq. 47)) then
call PRINT(PAGE,23,2,'Enter the preset count for this counter: *
* ',ATRIB6,46)
FLDCOL=43
elseif (ELETYP .eq. 42) then
call PRINT(PAGE,23,2,'Enter the preset time for this timer: *
* ',ATRIB6,43)

Appendix F. SYSTEM PROGRAM SOURCE LISTING
FLDCOL=40
endif

do J=1,5
  Q(J)=char(0)
continue
H=5
900  call KYBDIN.(true.,Q,5,H,23,FLDCOL,KEYCOD)
if ((KEYCOD .lt. 3).or.(KEYCOD .eq. 7).or.(KEYCOD .eq. 9)) then
  C  (cursor up,cursor down,carriage return,F2)
    do 40 J=1,5
      if (Q(J) .lt. LIMIT(J)) goto 150
      if (Q(J) .gt. LIMIT(J)) then
        call BEEP(23,FLDCOL)
        goto 800
      endif
    continue
  40  con(inue
  150  if (((KEYCOD .ne. 7) .and. (Q(5) .eq. char(0))) .or.
            (KEYCOD .eq. 9)) then
call BINASC(SBOX(ELMNTI(ELEMNT,3)>.3),Q(1),Q(2),Q(3),Q(4),Q(5))
else
call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),SBOX(ELMNTI(ELEMNT,3)>.3))
endif
call HVAL4(7,COL(COLUMN)+4,SBOXC(ELMNTI(ELEMNT,3)>.3),5,Q)
if (((KEYCOD .eq. 7) .and. (ELETYP .ne. 47)) .or.
     (KEYCOD .eq. 9)) then
  call PROMPT(KEY)
  return
elseif ((KEYCOD .eq. 1) .and. (ELETYP .eq. 47)) then
  goto 250
else
  goto 300
endif
elseif (KEYCOD .eq. 8) then
  call BEEP(23,FLDCOL)
  goto 900
endif
C - - - - - - - - - enter zero coil reference - - - - - - - - -
250  call LOCATE(PAGE,9,COL(COLUMN)+4)
if (SBOX(ELMNTI(ELEMNT,3)>.3) .gt. 0) then
call BINASC(SBOX(ELMNTI(ELEMNT,3)>.3),Q(1),Q(2),Q(3),Q(4),Q(5))
call HRCAT(PAGE,SBOXC(ELMNTI(ELEMNT,3)>.3),ATRIB3,1)
  do 50 J=2,5
    if (Q(J) .ne. char(0)) then
      call HRCAT(PAGE,Q(J),ATRIB3,1)
    endif
  50  continue
else
  call HRCAT(PAGE,' ',ATRIB3,5)
endif
251  call ERASLN(PAGE,23,ATRIB6)
call CSRFF
  call PRINT(PAGE,23,2,'Press Y or C to select the coil type (press
*<J to delete the coil)',ATRIB6,67)
252  call KYBDIN.(false.,Q,1,H,24,15,KEYCOD)
if (KEYCOD .eq. 12) then
  call LOCATE(PAGE,9,COL(COLUMN)+4)
call HRCARI(PAGE,'Y',1)
  SBOXC(ELMNTI(ELEMNT,3))='Y'
  XYFLAG=.true.
  K=1023
elseif (KEYCOD .eq. 13) then
  call LOCATE(PAGE,9,COL(COLUMN)+4)
call HRCARI(PAGE,'C',1)
  SBOXC(ELMNTI(ELEMNT,3))='C'
  XYFLAG=.false.
  K=511

Appendix F. SYSTEM PROGRAM SOURCE LISTING
elseif ((KEYCOD.lt.3).or.(KEYCO0.eq.7).or.(KEYCOD.eq.9)) then
   do 51 J=1,5
      Q(J)=char(0)
   continue
else
call BEEP(23,57)
goto 252
dendif
call ERASLN(PAGE,23,ATRIB6)
call CSROFF
call PRINT(PAGE,23,2,'Enter the zero coil reference number for this
counter: ','ATRIB6,59)
do 60 J=1,5
   Q(J)=char(0)
60 continue
H=5
call KYBDIN(.true.,Q,4,H,23,57,KEYCOD)
call CSROFF
if (((KEYCOD .lt. 3).or.(KEYCOD .eq. 7).or.(KEYCOD .eq. 9)) then
   TEMP=SBOX(ELMNTI(ELEMNT,3)`3)
   if (((KEYCOD .ne. 7) .and. (Q(5) .eq. char(0))) .or.
      (KEYCOD .eq. 9)) then
      call BINASC(SBOX(ELMNTI(ELEMNT,3)`3),Q(1),Q(2),Q(3),Q(4),Q(5))
   else
      call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),TEMP)
   endif
   if (TEMP .gt. K) then
      C {value exceeds limit}
call BEEP(23,57)
goto 350
   endif
   if (TEMP .eq. 0) then
      if ((XYFLAG) .and. (TYPREF(TEMP) .eq. 'X')) then
         C {X type element not allowed}
call BEEP(23,57)
goto 350
      endif
   endif
   SBOXI(ELMNTI(ELEMNT,3)`3)=TEMP
call CSROFF
call LOCATEI(PAGE,9,Col(COLUMN)+4)
call WRCHATI(PAGE,' ',ATRIB4,5)
if (SBOXI(ELMNTI(ELEMNT,3)`3).gt. 0) then
   call LOCATEI(PAGE,9,COl(COLUMN)+4)
call WRCHATI(PAGE,SBOXI(ELMNTI(ELEMNT,3)),ATRIB4,1)
do 70 J=1,5
   if (Q(J) .ne. char(0)) then
      call WRCHATI(PAGE,Q(J),ATRIB4,1)
   endif
70 continue
dendif
either ((KEYCOD .eq. 7).or.(KEYCOD .eq. 9)) then
   call PROMPT(KEY)
   return
eelse (KEYCOD .eq. 1) then
   goto 700
eelse
goto 300
dendif
eelseif (KEYCOD .eq. 8) then
   goto 450
dendif
else

C -- large box --

650  call HIVAL3(2,26,LBOX(ELMNTI(ELEMNT,3),1),2)
    call ERASLN(PAGE,23,ATRIB6)

750  call CSROFF
    call PRINT(PAGE,23,2,'Enter the reference number for this drum:
      *',ATRIB6,44)
    do 80 J=1,5
      Q(J)=char(0)
  80    continue

H=5

850  call KYBDIN(.true.,Q,2,H,23,44,KEYCOD)
    if (((KEYCOD .lt. 8) .or. (KEYCOD .eq. 9)) then
      C (cursor up,down,left,right,s-tab,tab,carrige return,F2)
        TEMP=LBOX(ELMNTI(ELEMNT,3),1)
        if (((KEYCOD .eq. 7) .and. (Q(5).eq.char(0))) .or.
           (KEYCOD .eq. 9)) then
          call BINASCILQ(ELMNTI(ELEMNT,3),1),Q(1),Q(2),Q(3),Q(4),Q(5))
          else
            call ASCBINLQ(1),Q(2),Q(3),Q(4),Q(5),TEMP)
          endif
    endif
    if (TEMP .gt. 30) then
      C (value exceeds limit)
      call BEEP(23,44)
      goto 750
    elseif (((KEYCOD .eq. 7) .and. (TEMP .gt. 0)) then
      if (LBXREF(TEMP) .gt. 0) then
        C (reference number already assigned)
        call BEEP(23,44)
        goto 750
      endif
    endif
    LBXREF(LBOX(ELMNTI(ELEMNT,3),1))=0
    LBOX(ELMNTI(ELEMNT,3),1)=TEMP
    if (TEMP .gt. 0) LBXREF(TEMP)=ELEMNT
    call HIVAL4(2,26,LBOX(ELMNTI(ELEMNT,3),1),2,Q)
    if (KEYCOD .eq. 1) then
      STEP=16
    goto 710
    elseif (((KEYCOD .eq. 2) .or. (KEYCOD .eq. 7)) then
      goto 110
    elseif (((KEYCOD .eq. 3) .or. (KEYCOD .eq. 5)) then
      COIL=15
      goto 120
    elseif (((KEYCOD .eq. 4) .or. (KEYCOD .eq. 6)) then
      COIL=1
      goto 120
    elseif (KEYCOD .eq. 9) then
      call PROMPTIKEY
      return
    endif
    elseif (KEYCOD .eq. 8) then
      (esc)
    goto 750
    else
      call BEEP(23,44)
      goto 850
    endif

C -- enter reset step --

110  call HIVAL3(5,34,LBOX(ELMNTI(ELEMNT,3),2),2)
    call ERASLN(PAGE,23,ATRIB6)

210  call CSROFF
    call PRINT(PAGE,23,2,'Enter the reset step for this drum: ',
      *ATRIB6,38)
    do 90 J=1,5
      Q(J)=char(0)
  90    continue

H=5

310  call KYBDIN(.true.,Q,2,H,23,38,KEYCOD)

Appendix F. SYSTEM PROGRAM SOURCE LISTING 213
if ((KEYCOD .lt. 8) .or. (KEYCOD .eq. 9)) then
  \( \text{ TEMP} = \text{LBOX}(\text{ELMNT}(\text{ELEMNT},3)^2) \)
  if ((KEYCOD .ne. 7) .and. (Q(5) .eq. char(0))) .or.
(  (KEYCOD .eq. 9)) then
    call BINASC(LBOX(ELMNT(ELMNT,3),2),Q(1),Q(2),Q(3),Q(4),Q(5))
  else
    call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),TEMP)
  endif
if (TEMP .gt. 16) then
  \( \text{call BEEP}(23,38) \) goto 210
else
  LBOX(ELMNT(ELMNT,3),2)=TEMP
go to 210
endif
if (KEYCOD .eq. 1) then
  goto 650
elseif ((KEYCOD .eq. 2) .or. (KEYCOD .eq. 7)) then
  goto 410
elseif ((KEYCOD .eq. 3) .or. (KEYCOD .eq. 5)) then
  \( \text{COIL} = 15 \)
go to 120
elseif ((KEYCOD .eq. 4) .or. (KEYCOD .eq. 6)) then
  \( \text{COIL} = 1 \)
go to 120
elseif (KEYCOD .eq. 9) then
  call PROMPTIKEY)
return
endif
elseif (KEYCOD .eq. 8) then
  \( \text{call BEEP}(23,38) \) goto 310
endif
C ————----—--- enter sec/count ————----—---
  call LOCATE(PAGE,4,33)
if (LBOX(ELMNT(ELMNT,3),3) .gt. 0) then
  call BINASC(LBOX(ELMNT(ELMNT,3),3),Q(1),Q(2),Q(3),Q(4),Q(5))
do 15 J=1,5
  if (Q(J) .eq. char(0)) then
    if (J .eq. 1) goto 15
    Q(J)="0"
  endif
  call WRCHAT(PAGE,Q(J),ATRIB3,1)
if (J .eq. 2) call WRCHAT(PAGE,'.',ATRIB3,1)
15 continue
else
  call WRCHAT(PAGE,' ',ATRIB3,5)
endif
call ERASLN(PAGE,23,ATRIB6)
call CSROFF
  call PRINT(PAGE,23,2,'Enter the count duration for this drum (mill
*iseconds): ',ATRIB6,60)
do 25 J=1,5
  Q(J)=char(0)
25 continue
H=5
call KBDBIN(.true.,Q,5,H,23,57,KEYCOD)
if ((KEYCOD .lt. 8) .or. (KEYCOD .eq. 9)) then
  \( \text{ CURSOR} = \text{LBOX}(\text{ELMNT}(\text{ELEMNT},3)^2) \)
do 35 J=1,5
  if (Q(J) .lt. LIMIT(J)) goto 620
  if (Q(J) .gt. LIMIT(J)) then
    \( \text{call BEEP}(23,57) \)
  goto 510
else
  call BEEP(23,58)
do 650
  \( \text{call BEEP}(23,38) \) goto 410
endif
Appendix F. SYSTEM PROGRAM SOURCE LISTING 214
endif

35 continue

620 if (((KEYCOD .ne. 7) .and. (Q(5) .eq. char(0))) .or.  
    (KEYCOD =. 9)) then
  call BINASCILBOX(ELMNTI(ELEMNT>3),3),Q(1),Q(2),Q(3),Q(4),Q(5))
else
  call ASCBIN(IQ(1),Q(2),Q(3),Q(4),Q(5),LBOX(ELMNTI(ELEMNT,3),3))
endif

45 continue

endif

if (KEYCOD .eq. 1) then
  goto 110
elseif ((KEYCOD .eq. 2) .or. (KEYCOD .eq. 7)) then
  STEP=1
  goto 710
elseif ((KEYCOD .eq. 5) .or. (KEYCOD .eq. 5)) then
  COIL=15
  goto 120
elseif (KEYCOD .eq. 4) .or. (KEYCOD .eq. 6)) then
  COIL=1
  goto 120
elseif (KEYCOD .eq. 9) then
  call PROMPT(KEY)
  return
endif

elseif (KEYCOD .eq. 8) then
  C {esc)
  goto 510
else
  call BEEP(23,57)
  goto 610
endif

C -- —---———---- enter count/step —— ---———----

710 call ERASLN(PAGE,23,ATRl6)

810 call HIVAL1(STEP+5,30,LBOX(ELMNTI(ELEMNT,3),STEP+3),5)

910 call KYBDIN(.true.,Q,5,H,23,50,KEYCOD)
if ((((KEYCOD .lt. 8) .or. (KEYCOD .eq. 9)) then
  (cursor up,down,left,right,s—tab,tab,carriage return,F2)
  do 65 J=1,5
    if (Q(J) .lt. LIMIT(J)) goto 720
    if (Q(J) .gt. LIMIT(J)) then
      call BEEP(23,50)
    goto 810
  endif
  H=5
endif

65 continue

720 call HIVAL2(STEP+5,30,LBOX(ELMNTI(ELEMNT,3),STEP+3),5,Q,KEYCOD)
if ((KEYCOD .eq. 1) then
  if (STEP .eq. 1) then
    goto 410
else
STEP=STEP-1
  goto 810
endif
elseif ((KEYCOD .eq. 2) .or. (KEYCOD .eq. 7)) then
  if (STEP .eq. 16) then
    goto 650
  else
    STEP=STEP+1
    goto 810
  endif
elseif ((KEYCOD .eq. 3) .or. (KEYCOD .eq. 5)) then
  COIL=15
  goto 420
elseif ((KEYCOD .eq. 4) .or. (KEYCOD .eq. 6)) then
  COIL=1
  goto 420
elseif (KEYCOD .eq. 9) then
  call PROMPT(KEY)
  return
endif
elseif (KEYCOD .eq. 8) then
  C {esc}
  goto 810
else
  call BEEP(23,50)
  goto 910
endif

C -----——--- enter output coil reference —--—-—--—-—120 call H1COIL(38+COIL*2,LBOXC(ELMNTI(ELEMNT,3)>COIL+l9)
x LBOXC(ELMNTI(ELEMNT,3),COIL);ATRIB3)
121 call ERASLN(PAGE,23,ATRIB6)
call CSROFF
call PRINT(PAGE,23,2,'Press Y or C to select the coil type (press
*<—J to delete the coil)',ATRIB6,67)
122 call KYBDIN(.false.,Q;1,H,23,57,KEYCOD)
if (KEYCOD .eq. 12) then
  call LOCATE(PAGE,1,39+COIL*2)
call MRCHAR(PAGE,'Y',1)
  LBOXC(ELMNTI(ELEMNT,3),COIL)='Y'
  XYFLAG=.true.
  K=1023
elseif (KEYCOD .eq. 13) then
  call LOCATE(PAGE,1,39+COIL*2)
call MRCHAR(PAGE,'C',1)
  LBOXC(ELMNTI(ELEMNT,3),COIL)='C'
  XYFLAG=.false.
  K=511
elseif ((KEYCOD.lt.8) .or. (KEYCOD.eq.9)) then
  do 74 J=1,5
    Q(J)=char(0)
  74 continue
  H=5
  if (XYFLAG) then
    call ERASLN(PAGE,23,ATRIB6)
call CSROFF
call PRINT(PAGE,23,2,'Enter the output coil reference number for t
this drum: ',';ATRIB6,58)
  do 75 J=1,5
    Q(J)=char(0)
  75 continue
else
  call BEEP(23,57)
goto 122
endif
220 call CSROFF
call PRINT(PAGE,23,2,'Enter the output coil reference number for t
his drum: ',;ATRIB6,58)
230 call KYBDIN(.true.,Q;4,H,23,56,KEYCOD)
321 if (((KEYCOD .lt. 8) .or. (KEYCOD .eq. 9)) then
  C {cursor up,down,left,right,carriage return,F2}
  TEMP=LBOXC(ELMNTI(ELEMNT,3),COIL+l9)
if (((KEYCOD .ne. 7) .and. (Q(5).eq.char(0)) .or.
(KEYCOD .eq. 9)) then
   call BINASC(LBOX(ELMNTI(ELEMNT,3),COIL+19),Q(1),Q(2),
   Q(3),Q(4),Q(5))
else
   call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),TEMP)
endif
if (TEMP .gt. K) then
   (value exceeds limit)
   call BEEP(23,56)
   goto 220
endif
if (TEMP .eq. 0) then
   LBOXC(ELMNTI(ELEMNT,3),COIL)= ' '
else
   if ((XYFLAG) .and. (TYPREF(TEMP) .eq. 'X')) then
      (X type element not allowed)
      call BEEP(23,57)
      goto 220
   endif
endif
LBOX(ELMNTI(ELEMNT;3),COIL+19)=TEMP
call CSROFF
call H2COIL(38+COIL*2,LBOX(ELMNTI(ELEMNT,3)»COIL+19)»
   * LBOXC(ELMNTI(ELEMNT,3)>COIL)»G)
if (KEYCOD .eq. 1) then
   STEP=16
   goto 420
elseif ((KEYCOD .eq. 2) .or. (KEYCOD .eq. 7)) then
   STEP=1
   goto 420
elseif ((KEYCOD .eq. 3) .or. (KEYCOD .eq. 5)) then
   if (COIL .eq. 1) then
      goto 650
   else
      COIL=COIL•1
      goto 120
   endif
elseif ((KEYCOD .eq. 4) .or. (KEYCOD .eq. 6)) then
   if (COIL .eq. 15) then
      goto 650
   else
      COIL=COIL+1
      goto 120
   endif
elseif (KEYCOD .eq. 9) then
   call PROMPTIKEY)
   return
endif
elseif (KEYCOD .eq. 8) then
   (esc)
   goto 121
else
   call BEEP(23,56)
   goto 320
endif
C - - - - - - - - - enter output coil sequence bits - - - - - - - - - -
420 call ERASLN(PAGE,23,ATRIB6)
call PRINT(PAGE,23,2,'Press the space bar to toggle ON or OFF the
   #output coil for this step (* = ON)'
520 call HISTEP(ATRIB3)
call HICOIL(38+COIL*2,LBOX(ELMNTI(ELEMNT,3),COIL+19),
   * LBOXC(ELMNTI(ELEMNT,3),COIL),ATRIB3)
call HIBIT(LBOX(ELMNTI(ELEMNT,3),STEP+34),ATRIB3)
820 call KYBDIN(,false,0,1,H,24,76,KEYCOD)
if (KEYCOD .lt. 7) then
   call HIBIT(LBOX(ELMNTI(ELEMNT,3),STEP+34),ATRIB4)
if (KEYCOD .eq. 1) then
   call HISTEP(ATRIB4)
if (STEP .eq. 1) then
C - - - - - - - - -
goto 120
else
   STEP=STEP-1
goto 520
endif
elseif (KEYCOD .eq. 2) then
   call HISTEP(ATRIB4)
   if (STEP .eq. 16) then
goto 120
else
   STEP=STEP+1
goto 520
endif
elseif ((KEYCOD .eq. 3) .or. (KEYCOD .eq. 5)) then
   call H1COIL(38+COIL*2,LBOX(ELMNTI(ELEMNT,3),COIL+19),
   LBOXC(ELMNTI(ELEMNT,3),COIL),ATRIB4)
   if (COIL .eq. 1) then
call HISTEP(ATRIB4)
goto 710
else
   COIL=COIL-1
goto 520
endif
elseif ((KEYCOD .eq. 4) .or. (KEYCOD .eq. 6)) then
   call H1COIL(38+COIL*2,LBOX(ELMNTI(ELEMNT,3),COIL+19),
   LBOXC(ELMNTI(ELEMNT,3),COIL),ATRIB4)
   if (COIL .eq. 15) then
call HISTEP(ATRIB4)
goto 710
else
   COIL=COIL+1
goto 520
endif
elseif ((KEYCOD .eq. 7) .or. (KEYCOD .eq. 10)) then
   call LOCATE(PAGE,STEP+5,39+COIL*2)
   if (AND16(LBOX(ELMNTI(ELEMNT,3),STEP+34),2**(COIL-1)).eq.
   2**(COIL-1)) then
call NRCHAT(PAGE,' ',ATRIB3,1)
   LBOXIELMNTI(ELEMNT,3),STEP+34)=
   OR16(LBOX(ELMNTI(ELEMNT,3),STEP+34),2**(COIL-1))
else
   call HRCHAT(PAGE,'¤',ATRIB3,1)
   LBOXIELMNTI(ELEMNT,3),STEP+34)=
   AND16(LBOX(ELMNTI(ELEMNT,3),STEP+34),(32767-2**(COIL-1)))
endif
   goto 820
elseif (KEYCOD .eq. 9) then
   call HIBIT(LBOX(ELMNTI(ELEMNT,3),STEP+34),ATRIB4)
call HISTEP(ATRIB4)
call H1COIL(38+COIL*2,LBOX(ELMNTI(ELEMNT,3),COIL+19),
   LBOXC(ELMNTI(ELEMNT,3),COIL),ATRIB4)
call PROMPT(KEY)
return
else
   call BEEP(1,1)
goto 820
endif
endif
call PROMPT(KEY)
return
end

SUBROUTINE HIVAL1(ROM,COL,VALUE,LENGTH)

C (right justifies and highlights value)

$INCLUDE:'QATRIB.BLK'

Appendix F. SYSTEM PROGRAM SOURCE LISTING 218
SUBROUTINE HIVAL2(ROW,COL,VALUE,LENGTH,G,KEYCOD)

C {right justifies and unhighlights value}
$INCLUDE:'QATRIB.BLK'

character*1 Q(5)
integer*2 COL, J, KEYCOD, LENGTH, ROW, VALUE

if (((KEYCOD .ne. 7) .and. (Q(5) .eq. char(0))) .or.
    (KEYCOD .eq. 9)) then
   call BINASC(VALUE,Q(1),Q(2),Q(3),Q(4),Q(5))
else
   call ASCBIN(Q(1),Q(2),Q(3),Q(4),Q(5),VALUE)
endif

end

SUBROUTINE HIVAL3(ROW,COL,VALUE,LENGTH)

C {left justifies and highlights value}
$INCLUDE:'QATRIB.BLK'

character*1 Q(5)
integer*2 COL, J, LENGTH, ROW, VALUE

call LOCATE(PAGE,ROW,COL)
if (VALUE .gt. 0) then
   do 10 J=6-LENGTH,5
      if (Q(J) .eq. char(0)) Q(J)= ' '
   call WRCHAT(PAGE,Q(J),ATRIB3,1)
   continue
else
   call WRCHAT(PAGE,' ',ATRIB3,LENGTH)
endif

end
SUBROUTINE HIVAL4(ROW,COL,VALUE,LENGTH,Q)
C {left justifies and unhighlights value)
$INCLUDE:'QATRIB.BLK'

character*1 Q(5)
integer*2 COL, J, LENGTH, ROW, VALUE

call CSROFF
call LOCATE(PAGE,ROW,COL)
call WRCHAR(PAGE,' ',ATTRIB4,LENGTH)
if (VALUE .gt. 0) then
  call LOCATE(PAGE,ROW,COL)
do 10 J=6-LENGTH,5
  if (Q(J) .ne. char(0)) then
    call WRCHAR(PAGE,Q(J),ATTRIB4,1)
  endif
10 continue
endif
return
end

SUBROUTINE HICOIL(COL,VALUE,TYPE,ATTRIB)
$INCLUDE:'QATRIB.BLK'
$INCLUDE:'QBOXES.BLK'

character*1 Q(5), TYPE
integer*2 ATTRIB, COL, J, K, VALUE

call LOCATE(PAGE,1,COL)
if (VALUE .gt. 0) then
  call WRCHAR(PAGE,'-',ATTRIB,1)
call WRCHAR(PAGE,TYPE,ATTRIB,1)
call WRCHAR(PAGE,'·',ATTRIB,1)
call BINASC(VALUE,Q(1),Q(2),Q(3),Q(4),Q(5))
K=1
do 10 J=2,5
  if (Q(J) .ne. char(0)) then
    K=K+1
    call LOCATE(PAGE,K,COL)
    if (K .eq. 5) then
      call WRCHAR(PAGE,'-',ATTRIB,1)
    else
      call WRCHAR(PAGE,' ',ATTRIB,1)
    endif
    call WRCHAR(PAGE,Q(J),ATTRIB,1)
    if (K .eq. 5) then
      call WRCHAR(PAGE,'-',ATTRIB,1)
    else
      call WRCHAR(PAGE,' ',ATTRIB,1)
    endif
  endif
10 continue
else
  call WRCHAR(PAGE,'-',ATTRIB,3)
do 20 J=2,4
    call LOCATE(PAGE,J,COL)
call WRCHAR(PAGE,' ',ATTRIB,3)
20 continue

SUBROUTINE H2COIL(COL, VALUE, TYPE, Q)

$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QBOXES.BLK'

character*1 Q(5), TYPE
integer*2 COL, J, K, VALUE

call LOCATE(PAGE,1,COL)
call WRCHAT(PAGE,'-',ATRIB,3)
do 10 J=2,4
   call LOCATE(PAGE,J,COL)
call WRCHAT(PAGE,' ',ATRIB,3)
10 continue
call LOCATE(PAGE,5,COL)
call WRCHAT(PAGE,'-',ATRIB,3)
if (VALUE .gt. 0) then
   call LOCATE(PAGE,1,COL+1)
call WRCHAT(PAGE,TYPE,ATRIB,1)
call BINASC(VALUE,Q(1),Q(2),Q(3),Q(4),Q(5))
   K=1
   do 20 J=1,5
      if (Q(J) .ne. char(0)) then
         K=K+1
         call LOCATE(PAGE,K,COL+1)
call WRCHAT(PAGE,Q(J),ATRIB,1)
      endif
20 continue
endif
return
end

SUBROUTINE HIBIT(VALUE, ATTRIB)

$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QEDIT.BLK'

integer*2 AND16, ATTRIB, VALUE

call LOCATE(PAGE,STEP+5,38+COIL*2)
if (AND16(VALUE,2**(COIL-1)) .eq. 2**(COIL-1)) then
   call WRCHAT(PAGE,' ',ATTRIB,1)
call WRCHAT(PAGE,'*',ATTRIB,1)
call WRCHAT(PAGE,' ',ATTRIB,1)
else
   call WRCHAT(PAGE,' ',ATTRIB,3)
endif
return
end

SUBROUTINE HISTEP(ATTRIB)

$INCLUDE: 'QATRIB.BLK'
$INCLUDE: 'QEDIT.BLK'

character*1 Q(5)
integer*2 ATTRIB, J
SUBROUTINE PASS1

C {builds rung lists}

$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QCONVERT.BLK'

integer*2 COUNT, ELEMNT, I, J, K

do 20 I=1,155
do 10 J=1,7
   LIST(I,J)=0
 10 continue
20 continue
do 70 RUNG=1,7
   RECORD=1
   SEGMNT=1
   NULRNG(RUNG)=.true.
   COLUMN=0
100 COLUMN=COLUMN+1
   if (ELEPTR(RUNG,COLUMN) .ne. 0) then
      NULRNG(RUNG)=.false.
   endif
   if (NODFLG(RUNG,COLUMN)) then
      C {add start rung segment node record to PASS 1 list}
      LIST(RECORD,RUNG,1)=-1
      LIST(RECORD,RUNG,2)=COLUMN
      LIST(RECORD,RUNG,3)=1
      LIST(RECORD,RUNG,4)=SEGMNT
      RECORD=RECORD+1
   endif
   C {add contact/horizontal/box record to PASS 1 list}
   if (ELEPTR(RUNG,COLUMN) .eq. 32767) then
      LIST(RECORD,RUNG,1)=1
      LIST(RECORD,RUNG,2)=0
      LIST(RECORD,RUNG,3)=COLUMN
   else
      if (ELEPTR(RUNG,COLUMN) .gt. 0) then
         ELEMNT=ELEPTR(RUNG,COLUMN)
      else
         ELEMNT=-ELEPTR(RUNG,COLUMN)
      endif
      if (((ELMT(ELEMNT,1) .ge. char(21)) .and.
         (ELMT(ELEMNT,1) .lt. char(48))) then
         LIST(RECORD,RUNG,1)=-4
         LIST(RECORD,RUNG,3)=ichar(ELMT(ELEMNT,2))
         LIST(RECORD,RUNG,4)=COLUMN
      endif
      if (ELMT(ELEMNT,1) .lt. char(41)) then
         LIST(RECORD,RUNG,2)=0
      elseif (ELMT(ELEMNT,1) .lt. char(45)) then
         LIST(RECORD,RUNG,2)=1
      else
         LIST(RECORD,RUNG,2)=2
      endif
   endif
   if (RUNG .eq. ichar(ELMT(ELEMNT,2))) then
      RECORD=RECORD+1
   else
      goto 200

Appendix F. SYSTEM PROGRAM SOURCE LISTING
endif
endif

L1ST(RECORD,RUNG,1)=1
L1ST(RECORD,RUNG,2)=ELEPTR(RUNG,COLUMN)
L1ST(RECORD,RUNG,3)=COLUMN
COUNT=0
do 30 K=1,RUNG-1
   if (.not. NULRNG(K)) COUNT=COUNT+1
   continue
   COUNT=COUNT+ABSRNG
   if (ELMNTC(ELEMNT,1) .gt. char(48)) then
      if (LBOX(ELEMNT,ELEMNT,3),1) .eq. 0) then
         write(2,1001) NTNUM,COUNT,COLUMN
         ERRNUM=ERRNUM+1
      endif
   elseif (ELMNTC(ELEMNT,1) .gt. char(21)) then
      if (SBOX(ELEMNT,ELEMNT,3),1) .eq. 0) then
         write(2,1001) NTNUM,COUNT,COLUMN
         ERRNUM=ERRNUM+1
      endif
   else
      if (ELMNTI(ELEMNT,3) .eq. 0) then
         write(2,1001) NTNUM,COUNT,COLUMN
         ERRNUM=ERRNUM+1
      endif
      if (ELMNTC(ELEMNT,1) .ge. char(48)) then
         COLUMN=COLUMN+8
      elseif (ELMNTC(ELEMNT,1) .ge. char(21)) then
         COLUMN=COLUMN+2
      endifendif

   if (NODFLG(RUNG,COLUMN+1)) then
      LIST(RECORD,RUNG,1)=-1
      LIST(RECORD,RUNG,2)=COLUMN+1
      LIST(RECORD,RUNG,4)=SEGMNT
   else
      endif
   endif
200 RECORD=RECORD+1

C  (add end rung segment node or passthru node record to PASS 1 list)
LIST(RECORD,RUNG,1)=-1
LIST(RECORD,RUNG,2)=COLUMN
LIST(RECORD,RUNG,4)=SEGMNT
if (NODFLG(RUNG,COLUMN+1)) then
   C (end rung segment node record)
   LIST(RECORD,RUNG,3)=-1
   SEGMNT=SEGMNT+1
else
   C (pass through node record)
   LIST(RECORD,RUNG,3)=0
endif
200 RECORD=RECORD+1
else
   if (RUNG .eq. 1) then
      write(2,1000) ABSRNG,NTNUM
      ERRNUM=ERRNUM+1
   endif
   endif
70 continue
return
1000 format('0','** ERROR ** Missing element on Rung ',*,I4,' in Network ',*,I4)
1001 format('0','** ERROR ** Element missing reference no. - Network ',*,I4,' Rung ',*,I4,' Column ',*,I2)
2000 format(1x), ' RECORD ',*,I2,' - NODE ',*,I2,'; start rung segment ',*,I2)
3000 format(1x), ' RECORD ',*,I2,' - NODE ',*,I2,' pass through node on segment ',*,I2)
4000 format(1x), ' RECORD ',*,I2,' - NODE ',*,I2,'; end rung segment ',*,I2)
5000 format(1x), ' RECORD ',*,I2,' - CONTACT ',*,I4,' on rung segment ',*,I2)
5001 format(1x), ' RECORD ',*,I2,' - CONTACT NOT ',*,I4,' on rung segment ',*,I2)
SUBROUTINE PASS2

C (decomposes rungs into rung segments, finds root rungs)
C (for each segment, and inserts start and end nodes for)
C (each segment into its root rung)

$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QCONVERT.BLK'

integer*2 COUNT, J, K, LASTEL, NODE1, NODE2, R, REC, ROOT1,
* ROOT2, SEG1, SEG2

logical*2 SHORTC, STRFLG

do 110 RUNG=1,2,1
RECORD=0
100 STRFLG=.false.
do 200 RECORD=RECORD+1
if (L1ST(RECORD,RUNG,1) .ne. 0) then
  COUNT=0
do 21 K=1,RUNG-1
if (.not. NULRNG(K)) COUNT=COUNT+1
21 continue
COUNT=COUNT+ABSRNG
write(2,9000) NTNNUM,COUNT,LIST(RECORD,RUNG,3)
ERRNUM=ERRNUM+1
endif

C (find root rung & root segment for the start rung segment node)
if (L1ST(RECORD,RUNG,1) .eq. 1) then
  STRFLG=.true.
  NODE1=LIST(RECORD,RUNG,2)
  if (NODE1 .eq. 1) then
    ROOT1=R
    SEG1=1
  else
    ROOT1=1
    SEG1=1
  endif
else
  do 10 R=RUNG,1,-1
if (.not. NODFLG(R,NODE1)) then
  ROOT1=R
  goto 300
10 continue
endif
C (find root segment)
SEG1=0
300 do 20 REC=1,155
if ((LIST(REC,ROOT1,1) .eq. 1) .and.
* (LIST(REC,ROOT1,2) .eq. NODE1)) then
  SEG1=LIST(REC,ROOT1,4)
  goto 400
endif
20 continue
COUNT=0
do 22 K=1,RUNG-1
if (.not. NULRNG(K)) COUNT=COUNT+1
continue
COUNT=COUNT+ABSRNG
write(2,9000) NTNNUM,COUNT,NODE1
ERRNUM=ERRNUM+1
go to 110
endif
else
  goto 200
endif

C -----——------—--—------—-----------C {find root rung & root segment for the end rung segment node)
400  LASTEL=0
      ROOT2=0
      NODE2=0
500  RECORD=RECORD+1
if (RECORD .gt. 155) then
  COUNT=0
  do 23 K=1,RUNG-1
    if (.not. NULRNG(K)) COUNT=COUNT+1
  23 continue
  COUNT=COUNT+ABSRNG
  write(2,1100) NTNNUM,COUNT,LASTEL
  ERRNUM=ERRNUM+1
  goto 110
endif
C {check for short circuit}
      if ((LIST(RECORD,RUNG,1) .eq. 1) .and.
*         (LIST(RECORD,RUNG,2) .ne. 0)) then
        LASTEL=LIST(RECORD,RUNG,3)
        goto 500
      endif
      if ((LIST(RECORD,RUNG,1) .eq. -1) .and.
*         (LIST(RECORD,RUNG,3) .eq. -1)) then
        NODE2=LIST(RECORD,RUNG,2)
C {find root rung}
      do 30 R=RUNG,1,-1
        if (.not. NODFLG(R,NODE2)) then
          ROOT2=R
        endif
        goto 600
  30 continue
C {find root segment}
600  SEG2=0
if (ROOT2 .gt. 0) then
  do 40 REC=1,155
    if ((LIST(REC,ROOT2,1) .eq. -1) .and.
*       (LIST(REC,ROOT2,2) .eq. NODE2)) then
      SEG2=LIST(REC,ROOT2,4)
      goto 700
    endif
  40 continue
COUNT=0
  do 24 K=1,RUNG-1
    if (.not. NULRNG(K)) COUNT=COUNT+1
  24 continue
COUNT=COUNT+ABSRNG
write(2,1100) NTNNUM,COUNT,NODE2-1
ERRNUM=ERRNUM+1
goto 110
endif
elseif (LIST(RECORD,RUNG,1) .eq. -4) then
  if (LASTEL .eq. 0) then
    COUNT=0
    do 31 K=1,RUNG-1
      if (.not. NULRNG(K)) COUNT=COUNT+1
  31 continue
    COUNT=COUNT+ABSRNG
    write(2,1400) NTNNUM,COUNT,LIST(RECORD,RUNG,3)
  endif
endif
ERRNUM=ERRNUM+1
endif
goto 100
else
go to 500
endif

700 if (LASTEL .eq. 0) then
   COUNT=0
   do 25 K=1,RUNG-1
      if (.not. NULRNG(K)) COUNT=COUNT+1
      continue
   write(2,8000) NTNUM,COUNT,NODE1,NODE2-1
   ERNUM=ERRNUM+1
endif

C -------——-----------—~———--—---- - - -C {compare root rungs}
if (ROOT1 .gt. ROOT2) then
   ROOT2=ROOT1
   SEG2=0
   do 50 REC=1,155
      if ((LIST(REC,ROOT1,1) .eq. -1) .and.
         (LIST(REC,ROOT1,2) .eq. NODE2)) then
         SEG2=LIST(REC,ROOT1,4)
         go to 800
      endif
   continue
50 continue
800 if (SEG1 .ne. SEG2) then
   COUNT=0
   do 26 K=1,ROOT1-1
      if (.not. NULRNG(K)) COUNT=COUNT+1
      continue
   COUNT=COUNT+ABSRNG
   write(2,1200) NTNUM,COUNT,NODE1
   ERRNUM=ERRNUM+1
   go to 100
endif
elseif (ROOT2 .gt. ROOT1) then
   ROOT1=ROOT2
   SEG1=0
   do 60 REC=155,1,-1
      if ((LIST(REC,ROOT2,1) .eq. -1) .and.
         (LIST(REC,ROOT2,2) .eq. NODE1)) then
         SEG1=LIST(REC,ROOT2,4)
         go to 900
      endif
   continue
60 continue
900 if (SEG1 .ne. SEG2) then
   COUNT=0
   do 27 K=1,ROOT2-1
      if (.not. NULRNG(K)) COUNT=COUNT+1
      continue
   COUNT=COUNT+ABSRNG
   write(2,1300) NTNUM,COUNT,NODE2-1
   ERRNUM=ERRNUM+1
   go to 100
endif
C -------——-----------—~———--—---- - - -C {compare root rung segments}
elseif (SEG1 .ne. SEG2) then
   COUNT=0
   do 28 K=1,ROOT1-1
      if (.not. NULRNG(K)) COUNT=COUNT+1
      continue
   COUNT=COUNT+ABSRNG
   write(2,1300) NTNUM,COUNT,NODE1,NODE2-1
   ERRNUM=ERRNUM+1
   go to 100
endif
C ---—---—-----------——---—— - -----—--
C {insert end of parallel segment node}
do 70 REC=155,1,-1
    if (LIST(REC,ROOT1,1) .ne. 0) goto 910
    continue
910 if ((LIST(REC,ROOT1,1) .eq. -1) .and.
        * (LIST(REC,ROOT1,1) .eq. NODE2)) then
    do 80 K=1,4
        LIST(REC+2,ROOT1,K)=LIST(REC,ROOT1,K)
    continue
    REC=REC-1
    if (LIST(REC,ROOT1,1) .ne. 1) goto 920
    LIST(REC+2,ROOT1,1)=-2
    LIST(REC+2,ROOT1,2)=RUNG
    LIST(REC+2,ROOT1,3)=0
    LIST(REC+2,ROOT1,4)=0
    do 12 K=1,4
        LIST(REC+1,ROOT1,K)=LIST(REC,ROOT1,K)
    continue
else
    do 11 K=1,4
        LIST(REC+2,ROOT1,K)=LIST(REC,ROOT1,K)
    continue
    REC=REC-1
    if (REC .gt. 1) goto 910
endif
C --------—--——----------—-——--------
C {insert start of parallel segment node}
if (NODE1 .eq. NODE2-1) goto 950
930 REC=REC-1
if (REC .lt. 1) goto 100
if ((LIST(REC,ROOT1,1) .eq. -1) .and.
    * (LIST(REC,ROOT1,1) .eq. NODE1+1)) then
    do 13 K=1,4
        LIST(REC+1,ROOT1,K)=LIST(REC,ROOT1,K)
    continue
    REC=REC-1
    if (LIST(REC,ROOT1,1) .ne. 1) goto 940
    do 14 K=1,4
        LIST(REC+1,ROOT1,K)=LIST(REC,ROOT1,K)
    continue
else
    do 15 K=1,4
        LIST(REC+1,ROOT1,K)=LIST(REC,ROOT1,K)
    continue
    goto 930
endif
950 LIST(REC,ROOT1,1)=-3
LIST(REC,ROOT1,2)=RUNG
LIST(REC,ROOT1,3)=0
LIST(REC,ROOT1,4)=0
C (process next rung segment)
    goto 100
endif
110 continue
C (check for short circuits on root rungs)
do 130 K=1,7
    SHORTC=.false.
    RECORD=0
120 RECORD=RECORD+1
    if (LIST(RECORD,K,1) .ne. 0) then
        if (LIST(RECORD,K,1) .eq. -1) then
            if (SHORTC) then
                NODE2=LIST(RECORD,K,2)
            else
                NODE1=LIST(RECORD,K,2)
            endif.
        endif.
    endif.
Appendix F. SYSTEM PROGRAM SOURCE LISTING
elseif (LIST(RECORD,K,1) .eq. -3) then
  SHORTC=.true.
  NODE2=NODE1
elseif ((LIST(RECORD,K,1) .eq. -2) .and. SHORTC) then
  COUNT=0
  do 29 J=1,RUNG-1
  if (.not. NULRNG(J)) COUNT=COUNT+1
  continue
  COUNT=COUNT+ABSRNG
  write(2,8000) NTNUM,COUNT,NODE1,NODE2
  ERRNUM=ERRNUM+1
  SHORTC=.false.
elseif ((LIST(RECORD,K,1) .eq. 1) .and.
          (LIST(RECORD,K,2) .ne. 0)) then
  SHORTC=.false.
endif
goto 120
endif
goto 120
endif
130 continue
140 return

C -----·—-——------———--------—---—--————-—---—---------—---—--———---—---—-—-
8000 format('0','** ERROR ** Short circuit - Network ',I4,', Rung ',I4,', Column ',I2)
9000 format('0','** ERROR ** Element not connected on left - Network ',I4,', Rung ',I4,', Column ',I2)
1100 format('0','** ERROR ** Element not connected on right - Network ',I4,', Rung ',I4,', Column ',I2)
1200 format('0','** ERROR ** Reverse flow - Network ',I4,', Rung ',I4,', Column ',I2)
1300 format('0','** ERROR ** Reverse flow - Network ',I4,', Rung ',I4,', Column ',I2)
1400 format('0','** ERROR ** Short circuit on input to box - Network ',I4,', Rung ',I4,', Column ',I2)
C ------——-—-—----—---—-—-———————————————————————-——————————-
end

SUBROUTINE PASS3
C {produces boolean code for each network and writes the}
C {code to a disk file }

$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QCONVERT.BLK'
$INCLUDE:'QBOXES.BLK'

integer*2 ELEMNT, HOLD, K, REC(7), STACK(25), STKPTR, TEMP
logical*2 STRFLG

RECORD=1
RUNG=1
REC(1)=1
do 10 K=2,7
  REC(K)=0
10 continue
STKPTR=0
HOLD=0
STRFLG=.true.
100 if (LIST(RECORD,RUNG,1) .ne. 0) then
    if ((LIST(RECORD,RUNG,1) .eq. 1) .and.
          (LIST(RECORD,RUNG,2) .ne. 0)) then
      C (element or horizontal)
      if (HOLD .gt. 0) then
        if (ELMN(T(HOLD,1), .eq. char(5))) then
          write(1,1000) 'A','C',ELMN(T(HOLD,3)
          write(1,1002) ELMNTI(HOLD,3)
        else
          if (TYPF(ELMN(T(HOLD,3)) .eq. 'X')) then
            write(1,1000) 'A','C',ELMN(T(HOLD,3)
            write(1,1002) ELMNTI(HOLD,3)
          end if
        end if
      else
        if (TYPF(ELMN(T(HOLD,3)) .eq. 'X')) then
          write(1,1000) 'A','C',ELMN(T(HOLD,3)
          write(1,1002) ELMNTI(HOLD,3)
        end if
      end if
    endif
  endif
  goto 100
write(1,1000) 'A','A',ELMNTI(HOLD,3)
else
write(1,1000) 'A','B',ELMNTI(HOLD,3)
endif
write(1,1001) TYPREF(ELMNTI(HOLD,3)),ELMNTI(HOLD,3)
endif
HOLD=0
elseif (HOLD .lt. 0) then
if (ELMNTI(-HOLD,1) .eq. char(8)) then
write(1,1000) 'A','F',ELMNTI(-HOLD,3)
write(1,2001) ELMNTII(-HOLD,3)
else
if (TYPREF(ELMNTI(-HOLD,3)) .eq. 'X') then
write(1,1000) 'A','D',ELMNTI(-HOLD,3)
else
write(1,1000) 'A','E',ELMNTI(-HOLD,3)
endif
write(1,2000) TYPREF(ELMNTI(-HOLD,3)),ELMNTI(-HOLD,3)
endif
HOLD=0
endif
if (STRFLG) then
HOLD=LIST(RECORD,RUNG,2)
STRFLG=.false. 4
else
ELEMNT=LIST(RECORD,RUNG,2)
if (ELEMNT .gt. 0) then
if (ELMNTCI(ELEMNT,1) .lt. char(9)) then
if (ELMNTCI(ELEMNT,1) .eq. char(5)) then
write(1,1000) 'B','C',ELMNTI(ELEMNT,3)
write(1,3001) ELMNTII(ELEMNT,3)
else
if (TYPREF(ELMNTI(ELEMNT,3)) .eq. 'X') then
write(1,1000) 'B','B',ELMNTI(ELEMNT,3)
endif
write(1,3000) TYPREF(ELMNTI(ELEMNT,3)),
ELMNTI(ELEMNT,3)
endif
elseif (ELMNTC(ELEMNT,1) .lt. char(48)) then
do 20 K=1,2
if (SBOX(ELMNTI(ELEMNT,3),K) .eq. 0) then
write(2,1200) NTNUM,RUNG,LIST(RECORD,RUNG,3)
ERRNUM=ERRNUM+1
endif
continue
20
if (ELMNTC(ELEMNT,1) .eq. char(51)) then
write(1,1000) 'G',' ',SBOX(ELMNTI(ELEMNT,3),1)
write(1,7000) SBOX(ELMNTI(ELEMNT,3),1)
write(1,1000) ' ','SBOX(ELMNTI(ELEMNT,3),2)
write(1,*)
elseif (ELMNTC(ELEMNT,1) .eq. char(42)) then
write(1,1000) 'F',' ',SBOX(ELMNTI(ELEMNT,3),1)
write(1,7001) SBOX(ELMNTI(ELEMNT,3),1)
write(1,1000) ' ','SBOX(ELMNTI(ELEMNT,3),2)
write(1,*)
elseif (SBOX(ELMNTI(ELEMNT,3),3) .eq. 0) then
write(2,1200) NTNUM,RUNG,LIST(RECORD,RUNG,3)
ERRNUM=ERRNUM+1
endif
write(1,1000) 'H',' ',SBOX(ELMNTI(ELEMNT,3),1)
write(1,7002) SBOX(ELMNTI(ELEMNT,3),1)
write(1,1000) ' ','SBOX(ELMNTI(ELEMNT,3),2)
write(1,*)
write(1,1000) ' ','SBOX(ELMNTI(ELEMNT,3)),
SBOX(ELMNTI(ELEMNT,3))
endif

else

do 30 K=1,3
  if (LBOX(ELMNTI(ELEMNT,3),K) .eq. 0) then
    write(2,1300) NTNUM
    ERRNUM=ERRNUM+1
  endif
continue
write(1,1000) 'K',' ',LBOX (ELMNTI(ELEMNT,3),1)
write(1,8000) LBOX(ELMNTI(ELEMNT,3),1)
write(1,1000) 'L',' ',LBOX(ELMNTI(ELEMNT,3),2)
write (1,*)
write(1,1000) 'M',' ',LBOX(ELMNTI(ELEMNT,3),3)
write (1,*)
do 40 K=4,19
  write(1,1000) 'M',NTNUM
  write (1,*)
40 continue

do 50 K=20,34
  write(1,1000) 'M',LBOX(ELMNTI(ELEMNT,3),K)
  write (1,*)
50 continue

do 60 K=35,50
  write(1,1000) 'M',LBOX(ELMNTI(ELEMNT,3),K)
  write (1,*)
60 continue
endif

else
  if (ELMNTC(ELEMNT,I) .eq. char(8)) then
    write(1,1000) 'B','F',ELMNTI(ELEMNT,3)
    write(1,5000) ELMTI(ELEMNT,3)
  else
    if (TYPREF(ELMNTI(ELEMNT,3)) .eq. 'X') then
      write(1,1000) 'B','A',ELMNTI(ELEMNT,3)
    else
      write(1,1000) 'B','E',ELMNTI(ELEMNT,3)
    endif
    write(1,5000) TYPREF(ELMNTI(ELEMNT,3)),ELMNTI(ELEMNT,3)
  endif
endif

elseif (LIST(RECORD,RUNG,1) .eq. -1) then
  {node}
  if (LIST(RECORD,RUNG,3) .eq. -1) then
    {end of rung segment}
      if (HOLD .gt. 0) then
        if (ELMNTC(HOLD,1) .eq. char(5)) then
          write(1,1000) 'C','C',ELMNTI(HOLD,3)
          write(1,5001) ELMNTI(HOLD,3)
        else
          if (TYPREF(ELMNTI(HOLD,3)) .eq. 'X') then
            write(1,1000) 'C','A',ELMNTI(HOLD,3)
          else
            write(1,1000) 'C','B',ELMNTI(HOLD,3)
          endif
          write(1,5000) TYPREF(ELMNTI(HOLD,3)),ELMNTI(HOLD,3)
        endif
        HOLD=0
      elseif (HOLD .lt. 0) then
        if (ELMNTC(-HOLD,1) .eq. char(8)) then
          write(1,1000) 'C','F',ELMNTI(-HOLD,3)
          write(1,6001) ELMNTI(-HOLD,3)
        else
          if (TYPREF(ELMNTI(-HOLD,3)) .eq. 'X') then
            write(1,1000) 'C','D',ELMNTI(-HOLD,3)
          else
            write(1,1000) 'C','E',ELMNTI(-HOLD,3)
          endif
        endif
  endif

Appendix F. SYSTEM PROGRAM SOURCE LISTING 230
endif
write(1,6000) TYPREF(ELMNTI(-HOLD,3)),ELMNTI(-HOLD,3)
endif
HOLD=0
else
write(1,1000) 'E',' ',0
write(1,*') 'OR STR'
endif
RUNG=STACK(STKPTR)
STKPTR=STKPTR-1
if (RUNG .gt. 0) then
  (no further parallel rung segments pending)
  write(1,1000) 'D',' ',0
  write(1,*') 'AND STR'
else
  (more parallel rung segments pending)
  RUNG=-RUNG
endif
STRFLG=.false.
endif
elseif LIST(RECORD,RUNG,1) .eq. -2) then

C (end of parallel segment)
TEMP=RUNG
RUNG=STACK(STKPTR)
STKPTR=STKPTR-1
if (RUNG .lt. 0) then
  RUNG=-RUNG
  TEMP=TEMP
endif
STKPTR=STKPTR+1
STACK(STKPTR)=TEMP
STRFLG=.true.
elseif LIST(RECORD,RUNG,1) .eq. -3) then

C (start of parallel segment)
STKPTR=STKPTR+1
if (STRFLG) then
  STACK(STKPTR)=-LIST(RECORD,RUNG,2)
else
  STACK(STKPTR)=LIST(RECORD,RUNG,2)
endif
STRFLG=.true.
elseif LIST(RECORD,RUNG,1) .eq. -4) then

C (LIST(RECORD,RUNG,1)=-4 - input to small box)
if (LIST(RECORD,RUNG,2) .ne. 0) then
  if (RUNG .eq. LIST(RECORD,RUNG,3)) then
    STKPTR=STKPTR+1
    STACK(STKPTR)=RUNG
  endif
  if (RUNG .eq. LIST(RECORD,RUNG,2)) then
    STACK(STKPTR)=STACK(STKPTR)
    STKPTR=STKPTR-1
  else
    RUNG=RUNG+1
    STRFLG=.true.
  endif
endif
endif
REC(RUNG)=REC(RUNG)+1
RECORD=REC(RUNG)
goto 100
endif
do 70 ELEPTR=1,7
if (ELEPTR(RUNG,12) .gt. 0) then
  if (ELMNTI(ELEPTR(RUNG,12),1) .eq. char(11)) then
    write(1,1000) 'J','C',ELMNTI(ELEPTR(RUNG,12),3)
    write(1,9001) ELMNTI(ELEPTR(RUN(12),3)
  else
    if (TYPREF(ELMNTI(ELEPTR(RUNG,12),3)) .eq. 'X') then
      write(1,1000) 'J','A',ELMNTI(ELEPTR(RUNG,12),3)
  endif
endif
enddo 70

Appendix F. SYSTEM PROGRAM SOURCE LISTING
else
  write(1,1000) 'J', 'B', ELMNTI(ELEPTR(RUNG,12),3)
endif
end

* wri{e(1,9000) TYPREF(ELMNTI(ELEPTR(RUNG,12),3)),
  ELMNTI(ELEPTR(RUNG,12),3)
end
if (ELMNTI(ELEPTR(RUNG,12),3) .eq. 0) then
  write(2,1400) NTNUM, RUNG
  ERRNUM=ERRNUM+1
endif
delse (ELEPTR(RUNG,12) .lt. 0) then
  if (ELMNTI(ELEPTR(RUNG,12),1) .eq. char(14)) then
    write(1,1000) 'J', 'F', ELMNTI(ELEPTR(RUNG,12),3)
    write(1,1101) ELMNTI(ELEPTR(RUNG,12),3)
    if (ELMNTI(ELEPTR(RUNG,12),3) .eq. 0) then
      write(2,1400) NTNUM, RUNG
      ERRNUM=ERRNUM+1
    endif
  endif
  elseif (ELEPTR(RUNG,12) .lt. 0) then
    if (ELEPTR(RUNG,12) .lt. 0) then
      if (ELMNTI(ELEPTR(RUNG,12),3) .eq. 0) then
        write(2,1400) NTNUM, RUNG
        ERRNUM=ERRNUM+1
      endif
    endif
  endif
70 continue
return

1000 format(2A,I5)
1001 format(4X,'STR ',A,I5)
1002 format(4X,'STR ',C,I5)
2000 format(4X,'STR ',A,I5)
2001 format(4X,'STR ',C,I5)
3000 format(4X,'AND ',A,I5)
3001 format(4X,'AND ',C,I5)
4000 format(4X,'AND NOT ',A,I5)
4001 format(4X,'AND NOT ',C,I5)
5000 format(4X,'OR ',A,I5)
5001 format(4X,'OR ',C,I5)
6000 format(4X,'OR ',A,I5)
6001 format(4X,'OR ',C,I5)
7000 format(4X,'CTR ',I3)
7001 format(4X,'TMR ',I3)
7002 format(4X,'UDC ',I3)
8000 format(4X,'DRUM ',I2)
9000 format(4X,'OUT ',A,I5)
9001 format(4X,'OUT ',C,I5)
1100 format(4X,'OUT NOT ',A,I5)
1101 format(4X,'OUT NOT ',C,I5)
1200 format(0,'** ERROR ** Incomplete data for small box - Network ',
  ',I4',' Rung ',I4,',' Column ',I2)
1300 format(0,'** ERROR ** Incomplete data for large box - Network ',
  ',I4')
1400 format(0,'** ERROR ** Coil missing reference number - Network ',
  ',I4',' Rung ',I4)
end

SUBROUTINE DISBOL(FILSPC)
C (convert display data to boolean code and write Boolean file to disk)
$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

Appendix F. SYSTEM PROGRAM SOURCE LISTING 232
integer*2 ERRNUM, LIST(155,7,4), RECORD, SEGMENT

logical*2 NODFLG(7,12), NULRNG(7)

common /CONVRT/ ERRNUM, LIST, NODFLG, NULRNG, RECORD, SEGMENT

character*14 FILSPC

integer*2 ANDB, ELEMT, ELETYP, I, J, K, NNPTR

logical*2 ENDFLG

open(1, file=FILSPC, status='NEW', access='SEQUENTIAL')
open(2, file='prn')

NTNNUM=1
NNPTR=NNHEAD
ABSRNG=1
ERRNUM=0

100 do 20 I=1,7
   do 10 J=1,12
      ELEPTR(I,J)=0
      NODFLG(I,J)=.false.
   10 continue
   NODFLG(I,1)=.true.
20 continue
ENDFLG=.false.
ELEMNT=NETNRK(NNPTR,1)

200 if (ELEMT .eq. NETNRK(NNPTR,2)) ENDFLG=.true.
   ELETYP=ichar(ELMNTC(ELEMNT,1))
   RUNG=ichar(ELMNTC(ELEMNT,2))
   COLUMN=ichar(ELMNTC(ELEMNT,3))
   if (ELETYP .eq. 1) then
      ELEPTR(RUNG,COLUMN)=32767
      elseif (ELETYP .eq. 2) then
         continue
      elseif (ELETYP .ge. 41) then
         ELEPTR(RUNG,COLUMN)=ELEMT
         elseif (ELETYP .ge. 45) then
         ELEPTR(RUNG+1,COLUMN)=ELEMT
         else
         ELEPTR(RUNG,COLUMN)=ELEMT
      endif
   endif
   if (RUNG .gt. 1) then
      if (AND8(ELMNTC(ELEMNT,4),char(64)) .eq. 64)
         * NODFLG(RUNG,COLUMN)=.true.
      elseif (AND8(ELMNTC(ELEMNT,4),char(4)) .eq. 4)
         * NODFLG(RUNG,COLUMN+1)=.true.
   endif
   if (ENDFLG) then
      call PASS1
      call PASS2
      call PASS3
      if (NETNRK(NNPTR,5) .ne. -1) then
         ABSRNG=ABSRNG+NETNRK(NNPTR,3)
         NNPTR=NETNRK(NNPTR,5)
         NTNNUM=NTNNUM+1

Appendix F. SYSTEM PROGRAM SOURCE LISTING
goto 100
endif
else
  ELEMNT=ELMNTI(ELEMNT,2)
goto 200
endif

if (ERRNUM .gt. 0) then
  close(1,status='DELETE')
else
  close(1)
endif

WRITE(2,2000) ERRNUM
2000 format(1X,I2,' ERROR($) DETECTED')
close(2)

NTNPTR=1
ABSRNG=1
NTHNUM=1
return
end

SUBROUTINE SAVE(FILSPC)
  {save the current ladder to disk}
$INCLUDE:'QLADDER.BLK'
$INCLUDE:'QBOXES.BLK'

character*14 FILSPC

integer*2 COUNT; ELEMNT, I, J, NNFRST(512), NMLAST(512), NNPTR

open(1,file=FILSPC,status='NEW',access='SEQUENTIAL', *
   form='BINARY')

write(1) 1850949962,1919247457

write(1) char(4), ELETOT
COUNT=0
NNPTR=NNHEAD
100 ELEMNT=NETNRK(NNPTR,1)
  NNFRST(NNPTR)=COUNT+1
  write(1) char(4)
do 10 J=1,4
  write(1) ELMNTC(ELEMNT,J)
10 continue
write(1) ELMNTI(ELEMNT,3)
if (ELEMNT .eq. NETWRK(NNPTR,2)) then
  NMLAST(NNPTR)=COUNT
if (NETWRK(NNPTR,5).ne. -1) then
  NNPTR=NETWRK(NNPTR,5)
goto 100
endif
else
  ELEMNT=ELMNTI(ELEMNT,2)
goto 200
endif
if (COUNT .ne. ELETOT) then
  call BEEP(1,1)
call LOCATE(0,18,10)
write(*,*) 'ERROR WRITING ELEMENT DATA'
close(1)
return
C {write network data}
write(1) char(5), NTWTOT
COUNT=1
NWPTR=NWHEAD
300 write(1) char(5)
write(1) NWFRST(NWPTR), NWLAST(NWPTR), NETWRK(NWPTR,3)
if (NETWRK(NWPTR,5) .eq. -1) then
  if (COUNT .ne. NTWTOT) then
    call BEEP(1,1)
    call LOCATE(0,18,10)
    write(*,*) 'ERROR WRITING NETWORK DATA'
    close(1)
    return
  endif
  NWPTR=NETWRK(NWPTR,5)
  COUNT=COUNT+1
  goto 300
endif

C {write TYPREF data}
write(1) char(6)
do 20 J=1,1023
  write(1) TYPREF(J)
20 continue
C {write small box data}
write(1) char(7), SBXTOT
do 40 I=1,SBXTOT
  write(1) char(7)
do 30 J=1,3
    write(1) SBOX(I,J)
30 continue
write(1) SBOXC(I)
40 continue
write(1) char(7)
do 50 J=1,128
  write(1) SBXREF(J)
50 continue
C {write large box data}
write(1) char(8), LBXTOT
do 80 I=1,LBXTOT
  write(1) char(8)
do 60 J=1,50
    write(1) LBOX(I,J)
60 continue
do 70 J=1,15
    write(1) LBOXC(I,J)
70 continue
80 continue
write(1) char(8)
do 90 J=1,30
  write(1) LBXREF(J)
90 continue
C {write end of file check data}
write(1) 541152800, 1212368212
close(1)
return
end
C
SUBROUTINE LOA0(FILSPC)
C {load a new ladder from disk}
$INCLUDE:'QLADDER.BLK'
$INCLUDE: 'QBOXES.BLK'

Appendix F. SYSTEM PROGRAM SOURCE LISTING 235
character*1  RECTYP
character*14  FILSPC
integer*2  COUNT, ELEMNT, I, J
integer*4  CHECK1, CHECK2
call INITLL
open1, file=FILSPC, status='OLD', access='SEQUENTIAL',
* form='BINARY')

C (read beginning of file check data)
read(1)  CHECK1, CHECK2
if ((CHECK1 .ne. 1850949962) .and. (CHECK2 .ne. 1919247457)) then
  call BEEP(1,1)
  call ERRMSG(17,7)
  close(1)
  return
endif

C (read element data)
read(1)  RECTYP
if (RECTYP .ne. char(4)) then
  call BEEP(1,1)
  call ERRMSG(17,8)
  close(1)
  call INITLL
  return
endif
read(1)  ELETOT
ELFREE=ELETOT+1
if (ELETOT .ge. ELEMAX) ELMFULL=.true.
do 40 I=1, ELETOT
   read(1)  RECTYP
   if (RECTYP .ne. char(4)) then
     call BEEP(1,1)
     call ERRMSG(17,8)
     close(1)
     call INITLL
     return
   endif
   do 30 J=1,4
      read(1)  ELMNTC(I,J)
   30 continue
   read(1)  ELMNTI(I,3)
40 continue
ELMNTI(ELETOT,2)=-1
ELMNTI(ELFREE,1)=0

C (read network data)
read(1)  RECTYP
if (RECTYP .ne. char(5)) then
  call BEEP(1,1)
  call ERRMSG(17,8)
  close(1)
  call INITLL
  return
endif
read(1)  NTMTOT
NNTAIL=NTMTOT
NNTFREE=NTMTOT+1
if (NTMTOT .ge. NTNMAX) NNTFULL=.true.
do 20 I=1, NTMTOT
   read(1)  RECTYP
   if (RECTYP .ne. char(5)) then
     call BEEP(1,1)
     call ERRMSG(17,8)
     close(1)
   endif
20 continue

Appendix F. SYSTEM PROGRAM SOURCE LISTING  236
call INITLL
return
endif
d0 10 J=1,3
read(1) NETRK(I,J)
10 continue
20 continue
NETRK(NMTAIL,5)=-1
NETRK(NMFREE,4)=0
C (read TYPREF data)
read(1) RECTYP
if (RECTYP .ne. char(6)) then
   call BEEP(1,1)
   call ERRMSG(17,8)
   close(1)
   call INITLL
return
endif
do 50 J=1,1023
read(1) TYPREF(J)
50 continue
C (read small box data)
read(1) RECTYP
if (RECTYP .ne. char(7)) then
   call BEEP(1,1)
   call ERRMSG(17,8)
   close(1)
   call INITLL
return
endif
read(1) SBXTOT
do 70 I=1,SBXTOT
read(1) RECTYP
if (RECTYP .ne. char(7)) then
   call BEEP(1,1)
   call ERRMSG(17,8)
   close(1)
   call INITLL
return
endif
do 60 J=1,3
   read(1) SBOX(I,J)
60 continue
read(1) SBOXC(I)
70 continue
read(1) RECTYP
if (RECTYP .ne. char(7)) then
   call BEEP(1,1)
   call ERRMSG(17,8)
   close(1)
   call INITLL
return
endif
do 80 J=1,128
   read(1) SBXREF(J)
80 continue
C (read large box data)
read(1) RECTYP
if (RECTYP .ne. char(8)) then
   call BEEP(1,1)
   call ERRMSG(17,8)
   close(1)
   call INITLL
return
endif
read(1) LBXTOT
do 21 I=1,LBXTOT
    read(1) RECTYP
    if (RECTYP .ne. char(8)) then
        call BEEP(1,1)
        call ERRMSG(17,8)
        close(1)
        call INITLL
        return
    endif
do 90 J=1,50
    read(1) LBOX(I,J)
90     continue
do 11 J=1,15
    read(1) LBOXC(I,J)
11     continue
21     continue
read(1) RECTYP
    if (RECTYP .ne. char(8)) then
        call BEEP(1,1)
        call ERRMSG(17,8)
        close(1)
        call INITLL
        return
    endif
do 31 J=1,30
    read(1) LBXREF(J)
31     continue
C (read end of file check data)
read(1) CHECK1, CHECK2
    if ((CHECK1 .ne. 541152800) .or. (CHECK2 .ne. 1212368212)) then
        call BEEP(1,1)
        call ERRMSG(17,8)
    endif
close(1)
return
end

---end of file check data---

SUBROUTINE BOLDISIFILSPC
C (read Boolean file from disk and preprocess data)
$INCLUDE:'QLADDER.BLK'

character*1 BXDATC(15), FIELD1(100), FIELD2(100)
integer*2 BXDATI(49), COUNT, ELEINX(90,10), ELENST(100),
* ELLAST, FIELD3(100), INDEX, LBXPTR, SBXPTR
common /DSPLAY/ BXDATI, COUNT, ELEINX, ELENST, ELLAST, FIELD1,
* FIELD2, FIELD3, INDEX, LBXPTR, SBXPTR, BXDATC

character*1 DUMMY
character*14 FILSPC

integer*2 CNTR, ERRNO, J, NEST, NSTCNT(10), TEMP3
open(1, file=FILSPC, status='OLD', access='SEQUENTIAL')

call INITLL
INDEX=0
NTNHPTR=0
SBXPTR=1
LBXPTR=1
read(1,1000,iostat=ERRNO, end=500, err=600) FIELD1(1),FIELD2(1),
* FIELD3(1) 
C (start new network)
100 if (FIELD1(1) .eq. 'A') then 
NTMPT = NTMPT + 1 
NETWK(NTMPT,1) = INDEX + 1 
NEST = 1
CNTR = 1
do 10 J = 1,10 
NSTCNT(J) = 1 
10 continue 
NSTCNT(1) = 2 
ELENST(1) = 1 
ELEINX(1,1) = 1 
else 
call BEEP(1,1) 
write(*,*) 'ERROR - NETWORK DOES NOT BEGIN WITH STR' 
stop 
endif 
200 CNTR = CNTR + 1
if (CNTR .gt. 100) then 
call BEEP(1,1) 
write(*,*) 'VARIABLE CNTR EXCEEDS LIMIT (A)' 
stop 
endif 
read(1,1000,iostra=ERRNO, end=500, err=600) FIELD1(CNTR), FIELD2(CNTR), FIELD3(CNTR) 
if (FIELD1(CNTR) .eq. 'A') then 
C (STR)
NEST = NEST + 1 
if (NEST .gt. 10) then 
call BEEP(1,1) 
write(*,*) 'VARIABLE NEST EXCEEDS LIMIT' 
stop 
endif 
elseif ((FIELD1(CNTR) .eq. 'B').or.(FIELD1(CNTR) .eq. 'C').or. 
* (FIELD1(CNTR) .eq. 'I').or.(FIELD1(CNTR) .eq. 'K')) then 
C (AND, OR, O/S, or DRUM)
if (FIELD1(CNTR) .eq. 'K') then 
do 20 J = 1,18 
read(1,1000,iostra=ERRNO, end=500, err=600) DUMMY, DUMMY, BXDATT(J)
20 continue 
do 30 J = 1,15 
read(1,1000,iostra=ERRNO, end=500, err=600) DUMMY, BXDATT(J), BXDATT(J+18)
30 continue 
do 40 J = 34,49 
read(1,1000,iostra=ERRNO, end=500, err=600) DUMMY, DUMMY, BXDATT(J)
40 continue 
endif 
elseif ((FIELD1(CNTR) .eq. 'D').or.(FIELD1(CNTR) .eq. 'E').or. 
* (FIELD1(CNTR) .eq. 'F').or.(FIELD1(CNTR) .eq. 'G')) then 
C (AND STR, OR STR, TMR, or CTR)
NEST = NEST - 1 
if (FIELD1(CNTR) .eq. 'F').or.(FIELD1(CNTR) .eq. 'G')) 
* read(1,1000,iostra=ERRNO, end=500, err=600) DUMMY, DUMMY, BXDATT(1)
elseif (FIELD1(CNTR) .eq. 'H') then 
C (UDC)
NEST = NEST - 2 
do 50 J = 1,2 
read(1,1000,iostra=ERRNO, end=500, err=600) DUMMY, DUMMY, BXDATT(J)
50 continue 
BXDATC(1) = DUMMY
ELEINX(NSTCNT(NEST+1),1) = CNTR
NSTCNT(NEST+1) = NSTCNT(NEST+1)+1
elseif (FIELD1(CNTR) .eq. 'J') then
   (OUT)
   continue
else
   call BEEP(1,1)
   write(*,*), 'ERROR READING BOOLEAN FILE'
   stop
endif
ELENST(CNTR)=NEST
if (FIELD1(CNTR) .ne. 'J') then
   ELENXINST(CNTR,NEST)=CNTR
   NSTACK(NEST)=NSTACK(NEST)+1
   if (NSTACK(NEST) .gt. 90) then
      call BEEP(1,1)
      write(*,*), 'VARIABLE NSTACK(NEST) EXCEEDS LIMIT'
      stop
   endif
   goto 200
endif
300 CNTR=CNTR+1
   if (CNTR .gt. 100) then
      call BEEP(1,1)
      write(*,*), 'VARIABLE CNTR EXCEEDS LIMIT (B)'
      stop
   endif
   read(1,1000,iostat=ERRNO,end=400,err=600) FIELD1(CNTR),
   * FIELD2(CNTR),FIELD3(CNTR)
   if (FIELD1(CNTR) .eq. 'J') then
      ELENST(CNTR)=NEST
      goto 300
   endif
400 COUNT=CNTR-1
   call PASSB
   NETNRK(NMPTR,2)=ELLAST
   if (EOF(1)) then
      NTNTO=NTNTPTR
      NTNVEL=NMTAIL
      NTNFFREE=NTNTPTR+1
      NTNTPTR=1
      NETNRK(NMTAIL,5)=-1
      NETNRK(NTNFFREE,4)=0
      ETOT=INDEX
      ELFREE=INDEX+1
      ELMNT(ELLAST,2)=-1
      ELMNT(ELFREE,1)=0
      close(1)
      return
   endif
   FIELD1(1)=FIELD1(CNTR)
   FIELD2(1)=FIELD2(CNTR)
   FIELD3(1)=FIELD3(CNTR)
   goto 100
500 call BEEP(1,1)
   write(*,*), 'ERROR - PREMATURE END OF FILE'
   stop
600 call BEEP(1,1)
   write(*,*), 'ERROR READING BOOLEAN FILE - ERROR NUMBER',ERRNO
   stop
1000 format(2A1;I5)
end

SUBROUTINE PASSB
   (generate ladder display data)
character*1 LINK(7,12)

integer*2 ANDRON(10), ANDCOL(10), AND8, CNTR, J, K, L, M, MARK,
* NEST, NSTCNT(10), ORRON(10), ORCOL(10), OR8, TEMP

CNTR=0
do 65 J=1,7
   do 55 K=1,12
      LINK(J,K)=char(0)
   55 continue
65 continue

do 10 J=1,10
   NSTCNT(J)=1
10 continue

ANDRON(1)=1
ANDCOL(1)=l
ORRON(l)=2
ORCOL(l)=1
100 CNTR=CNTR+1
if (CNTR .gt. COUNT) then
   ELLAST=INDEX
   do 20 J=1,7
      RONCNT(J)=0
20 continue
   do 30 J=NETNRK(NTNPTR,3),INDEX
      RUNG=ichar(ELMNTC(J,3))
      ELMNTC(J,4)=LINK(RUNG,ichar(ELMNTC(J,3)))
      RONCNT(RUNG)=RONCNT(RUNG)+1
30 continue
   NETNRK(NTNPTR,3)=0
   do 40 J=1,7
      if (RONCNT(J) .gt. 0) NETNRK(NTNPTR,3)=NETNRK(NTNPTR,3)+1
40 continue
C (put link data into element data list and count number of rungs)
   ELLAST=INDEX
   do 20 J=1,7
      RONCNT(J)=0
20 continue
   do 30 J=NETNRK(NTNPTR,3),INDEX
      RUNG=ichar(ELMNTC(J,3))
      ELMNTC(J,4)=LINK(RUNG,ichar(ELMNTC(J,3)))
      RONCNT(RUNG)=RONCNT(RUNG)+1
30 continue
   NETNRK(NTNPTR,3)=0
   do 40 J=1,7
      if (RONCNT(J) .gt. 0) NETNRK(NTNPTR,3)=NETNRK(NTNPTR,3)+1
40 continue
C (add type 2 elements where there are type 12 right links)
   do 60 J=1,7
      do 50 K=1,11
         if (AND8(LINK(J,K),char(15)) .eq. 12) then
            INDEX=INDEX+1
            ELMNTC(INDEX,1)=char(2)
            ELMNTC(INDEX,2)=J
            ELMNTC(INDEX,3)=K
            ELMNTC(INDEX,4)=LINK(J,K)
            ELMNTC(INDEX,5)=MARK
            ELMNTC(INDEX,6)=INDEX
            ELMNTC(MARK,1)=INDEX
            ELMNTC(MARK,2)=INDEX
30 continue
   ELMNTK(ELLAST,2)=INDEX+1
   return
   endif
   INDEX=INDEX+1
   NEST=ELENST(CNTR)
   NSTCNT(NEST)=NSTCNT(NEST)+1
   if ((FIELD1(CNTR) .eq. 'A') .or. (FIELD1(CNTR) .eq. 'B') .or.
* (FIELD1(CNTR) .eq. 'C')) then
   C (STR, AND, OR)
      if (FIELD2(CNTR) .eq. 'A') then
         ELMNTC(INDEX,1)=char(3)
         TYPREF(FIELD3(CNTR))='X'
      elseif (FIELD2(CNTR) .eq. 'B') then
         ELMNTC(INDEX,1)=char(4)
      else
         ELMNTC(INDEX,1)=char(5)
   C (END)
TYPREF(FIELD3(CNTR))='Y'
elseif (FIELD2(CNTR) .eq. 'C') then
  ELMNTC(INDEX,1)=char(5)
endif

if (FIELDZICNTRI .eq. 'C') {hen
  ELMNTC(INDEX,1)=char(5)
elseif (FIELDZICNTR) .eq. 'D') {hen
  ELMNTC(INDEX,1)=char(6)
  TYPREF(FIELD3(CNTR))='X'
endif

if (FIELDZICNTR) .eq. 'E') {hen
  ELMNTC(INDEX,1)=char(7)
  TYPREF(FIELD3(CNTR))='Y'
elseif (FIELDZICNTR) .eq. 'F') {hen
  ELMNTC(INDEX,1)=char(8)
endif

ELMNTC(INDEX,3)=FIELD3(CNTR)

if (FIELD1(CNTR) .eq. 'A') then
  if (NEST .eq. 1) {hen
    ELMNTC(INDEX,2)=ANDRON(1)
    ELMNTC(INDEX,3)=ANDCOL(1)
    LINK(ANDRON(NEST),ANDCOL(NEST))=char(1)
    LINK(ANDRON(NEST),ANDCOL(NEST)+1)=char(16)
    ANDCOL(NEST)=ANDCOL(NEST)+1
  else
    K=NEST-1
    if (FIELD1(ELEINC(NSTCNT(K),K)) .eq. 'D') {hen
      ELMNTC(INDEX,2)=ANDRON(K)
      ELMNTC(INDEX,3)=ANDCOL(K)
      * OR8L LINK(ANDRON(K),ANDCOL(K)),char(16)
      LINK(ANDRON(K),ANDCOL(K)-1)=char(1)
      OR8L LINK(ANDRON(K),ANDCOL(K)-1),char(16)
      ANDRON(NEST)=ANDRON(K)
      ANDCOL(NEST)=ANDCOL(K)+1
      ORRON(NEST)=ANDRON(K)+1
      ORCOL(NEST)=ANDCOL(K)
    elseif (FIELD1(ELEINC(NSTCNT(K),K)) .eq. 'E') {hen
      ELMNTC(INDEX,2)=ORRON(K)
      ELMNTC(INDEX,3)=ORCOL(K)
      LINK(ORRON(K),ORCOL(K))=char(1)
      LINK(ORRON(K),ORCOL(K)+1)=char(16)
      if (ORCOL(K) .eq. 1) {hen
        LINK(ORRON(K),ORCOL(K))=char(1)
      else
        LINK(ORRON(K),ORCOL(K))=char(97)
        LINK(ORRON(K),ORCOL(K)-1)=char(1)
        LINK(ORRON(K),ORCOL(K)-1)=char(16)
        J=1
        200 TEMP=AND8(LINK(ORRON(K)-J,ORCOL(K)),char(128))
        if (TEMP .ne. 128) {hen
          LINK(ORRON(K)-J,ORCOL(K))=char(192)
        else
          J=J+1
          go to 200
        endif
      endif
      ANDRON(NEST)=ORRON(K)
      ANDCOL(NEST)=ORCOL(K)+1
      ORRON(NEST)=ORRON(K)+1
      ORCOL(NEST)=ORCOL(K)
    elseif (FIELD1(ELEINC(NSTCNT(K),K)) .eq. 'F') .or.
      (FIELD1(ELEINC(NSTCNT(K),K)) .eq. 'G') .or.
      (FIELD1(ELEINC(NSTCNT(K),K)) .eq. 'H') then
      ELMNTC(INDEX,2)=ORRON(K)
      ELMNTC(INDEX,3)=ORCOL(K)
      LINK(ORRON(K),ORCOL(K))=char(1)
      LINK(ORRON(K),ORCOL(K)+1)=char(16)
    endif
  endif
else (FIELD1(CNTR) .eq. 'A') then
  ELMNTC(INDEX,2)=ANDRON(K)
  ELMNTC(INDEX,3)=ANDCOL(K)
  LINK(ANDRON(K),ANDCOL(K))=char(1)
  LINK(ANDRON(K),ANDCOL(K)-1)=char(16)
  ANDRON(NEST)=ANDRON(K)+1
  ANDCOL(NEST)=ANDCOL(K)
  ORRON(NEST)=ANDRON(K)
  ORCOL(NEST)=ANDCOL(K)+1
endif
if (ANDCOL(K).gt. ORCOL(K)+1) then
  (horizontal element)
  INDEX=INDEX+1
  ELMNTC(INDEX,1)=char(1)
  ELMNTC(INDEX,2)=ORRON(K)
  ELMNTC(INDEX,3)=ORCOL(K)+1
  LINK(ORRON(K),ORCOL(K)+1)=
  * OR8(LINK(ORRON(K),ORCOL(K)+1),char(33))
  LINK(ORRON(K),ORCOL(K))=
  * OR8(LINK(ORRON(K),ORCOL(K)),char(2))
  LINK(ORRON(K),ORCOL(K)+2)=char(16)
endif
ANDRON(NEST)=ORRON(K)
ANDCOL(NEST)=ORCOL(K)+1
ORRON(NEST)=ORRON(K)+1
ORCOL(NEST)=ORCOL(K)
else
call BEEP(1,1)
write(*,*): 'CONVERSION ERROR - STR'
stop
dendif
endif
C -------*------- AND -------*-------
eifl (FIELD1(CNTR) .eq. 'B') then
  ELMNTC(INDEX,2)=ANDRON(NEST)
  ELMNTC(INDEX,3)=ANDCOL(NEST)
  LINK(ANDRON(NEST),ANDCOL(NEST))=
  * OR8(LINK(ANDRON(NEST),ANDCOL(NEST)),char(33))
  LINK(ANDRON(NEST),ANDCOL(NEST)-1)=
  * OR8(LINK(ANDRON(NEST),ANDCOL(NEST)-1),char(2))
  LINK(ANDRON(NEST),ANDCOL(NEST)+1)=char(16)
  ANDCOL(NEST)=ANDCOL(NEST)+1
C -------*------- OR -------*-------
eifl (FIELD1(CNTR) .eq. 'C') then
  ELMNTC(INDEX,2)=ORRON(NEST)
  ELMNTC(INDEX,3)=ORCOL(NEST)
  if (ORCOL(NEST) .eq. 1) then
    LINK(ORRON(NEST),ORCOL(NEST))=char(5)
    LINK(ORRON(NEST),ORCOL(NEST)+1)=char(80)
    LINK(ORRON(NEST)-1,ORCOL(NEST))=
    * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)),char(8))
    LINK(ORRON(NEST)-1,ORCOL(NEST)+1)=
    * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)+1),char(128))
    elseif (ANDCOL(NEST) .le. ORCOL(NEST)+1) then
      LINK(ORRON(NEST),ORCOL(NEST))=char(101)
      LINK(ORRON(NEST),ORCOL(NEST)-1)=
      * OR8(LINK(ORRON(NEST),ORCOL(NEST)-1),char(6))
      LINK(ORRON(NEST),ORCOL(NEST)+1)=char(80)
      LINK(ORRON(NEST)-1,ORCOL(NEST))=
      * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)),char(136))
      LINK(ORRON(NEST)-1,ORCOL(NEST)-1)=
      * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)-1),char(8))
      LINK(ORRON(NEST)-1,ORCOL(NEST)+1)=
      * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)+1),char(128))
      else
        LINK(ORRON(NEST),ORCOL(NEST))=
        * OR8(LINK(ORRON(NEST),ORCOL(NEST)),char(97))
        LINK(ORRON(NEST),ORCOL(NEST)-1)=
        * OR8(LINK(ORRON(NEST),ORCOL(NEST)-1),char(6))
        LINK(ORRON(NEST),ORCOL(NEST)+1)=char(16)
        LINK(ORRON(NEST)-1,ORCOL(NEST))=
        * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)),char(128))
        LINK(ORRON(NEST)-1,ORCOL(NEST)-1)=
        * OR8(LINK(ORRON(NEST)-1,ORCOL(NEST)-1),char(8))
        do 70 J=ORCOL(NEST)+1,ANDCOL(NEST)-1
          (horizontal element)
          INDEX=INDEX+1
          ELMNTC(INDEX,1)=char(1)
Appendix F. SYSTEM PROGRAM SOURCE LISTING
ELMNTC(INDEX,1)=ORRON(NEST)
ELMNTC(INDEX,2)=J
LINK(ORRON(NEST),J)=OR8(LINK(ORRON(NEST),J),char(33))
LINK(ORRON(NEST),J-1)=
  * OR8(LINK(ORRON(NEST),J-1),char(2))
LINK(ORRON(NEST),J+1)=char(16)
70 continue
LINK(ORRON(NEST),ANDCOL(NEST)-1)=
  * OR8(LINK(ORRON(NEST),ANDCOL(NEST)-1),char(4))
LINK(ORRON(NEST),ANDCOL(NEST))=char(64)
LINK(ORRON(NEST)-1,ANDCOL(NEST)-1)=
  * OR8(LINK(ORRON(NEST)-1,ANDCOL(NEST)-1),char(8))
LINK(ORRON(NEST)-1,ANDCOL(NEST))=
  * OR8(LINK(ORRON(NEST)-1,ANDCOL(NEST)),char(128))
ENDIF
ORRON(NEST)=ORRON(NEST)+1
ENDIF
C
elseif (FIELD1(CNTR) .eq. 'D') then
INDEX=INDEX+1
ANDRON(NEST)=ANDRON(NEST)+1
ORRON(NEST)=ORRON(NEST)+1
C
elseif (FIELD1(CNTR) .eq. 'E') then
INDEX=INDEX+1
K=NEST+1
if (ANDCOL(K) .ge. ANDCOL(NEST)) then
  if (ANDCOL(K) .lt. ANDCOL(NEST)) then
    L=K
    M=NEST
  else
    L=NEST
    M=K
  endif
  do 40 J=ANDCOL(L),ANDCOL(M)-1
  INDEX=INDEX+1
  ELMNTC(INDEX,1)=char(1)
  ELMNTC(INDEX,2)=ANDRON(L)
  ELMNTC(INDEX,3)=J
  LINK(ANDRON(L),J)=OR8(LINK(ANDRON(L),J),char(33))
  LINK(ANDRON(L),J-1)=OR8(LINK(ANDRON(L),J-1),char(2))
  LINK(ANDRON(L),J+1)=char(16)
  40 continue
  ANDCOL(L)=ANDCOL(M)
ENDIF
C
elseif (FIELD1(CNTR) .eq. 'F') or (FIELD1(CNTR) .eq. 'G') or
  (FIELD1(CNTR) .eq. 'H') then
  if (FIELD1(CNTR) .eq. 'H') then
    
Appendix F. SYSTEM PROGRAM SOURCE LISTING
K=1
else
K=0
endif
do 15 J=K,0,-1
if (ANDCOL(NEST+J) .lt. ANDCOL(NEST+J+1)) then
    (horizontal element)
    ELMNTC(INDEX,1)=char(1)
    ELMNTC(INDEX,2)=ANDRON(NEST)
    ELMNTC(INDEX,3)=ANDCOL(NEST)
    LINK(ANDRON(NEST),ANDCOL(NEST))=
*     OR8(LINK(ANDRON(NEST),ANDCOL(NEST)),char(33))
    LINK(ANDRON(NEST),ANDCOL(NEST)-1)=
*     OR8(LINK(ANDRON(NEST),ANDCOL(NEST)-1),char(2))
    LINK(ANDRON(NEST),ANDCOL(NEST)+1)=char(16)
    ANDCOL(NEST+J)=ANDCOL(NEST+J+1)
    INDEX=INDEX+1
endif
15 continue
END

ELMNTC(INDEX,3)=SBXPTR
SBXREF(FIELD1(CNTR))=INDEX
SBOX(SBXPTR,1)=FIELD1(CNTR)
SBOX(SBXPTR,2)=BXDATI(1)
if (FIELD1(CNTR) .eq. 'F') then
    ELMNTC(INDEX,1)=char(42)
elseif (FIELD1(CNTR) .eq. 'G') then
    ELMNTC(INDEX,1)=char(41)
else
    ELMNTC(INDEX,1)=char(47)
endif
SBXPTR=SBXPTR+1
SBXTOT=SBXTOT+1
ELMNTC(INDEX,2)=ANDRON(NEST)
ELMNTC(INDEX,3)=ANDCOL(NEST)
LINK(ANDRON(NEST),ANDCOL(NEST))=
*     OR8(LINK(ANDRON(NEST),ANDCOL(NEST)),char(33))
*     OR8(LINK(ANDRON(NEST),ANDCOL(NEST)-1),char(2))
    LINK(ANDRON(NEST)+1,ANDCOL(NEST)-1)=
*     OR8(LINK(ANDRON(NEST)+1,ANDCOL(NEST)-1),char(2))
if (FIELD1(CNTR) .eq. 'H')
*     LINK(ANDRON(NEST)+2,ANDCOL(NEST)-1)=
*     OR8(LINK(ANDRON(NEST)+2,ANDCOL(NEST)-1),char(2))
    LINK(ANDRON(NEST),ANDCOL(NEST)+3)=char(16)
    ANDCOL(NEST)=ANDCOL(NEST)+3
END
* OR8(LINK(ANDRON(NEST)), ANDCOL(NEST)-1), char(2))
  LINK(ANDRON(NEST)), ANDCOL(NEST)+1)=char(16)
  ANDCOL(NEST)=ANDCOL(NEST)+1
  INDEX=INDEX+1
endif
ELMNTC(INDEX,1)=char(50)
ELMNTC(INDEX,2)=ANDRON(NEST)
ELMNTC(INDEX,3)=ANDCOL(NEST)
ELMNTC(INDEX,3)=LBXPTR
LBXREF(FIELD3(CNTR))=INDEX
LBX(LBX PTR,1)=FIELD3(CNTR)
do 25 J=1,49
LBX(LBX PTR,J+1)=BXDAT(J)
25 con tinue
do 35 J=1,15
LBX PTR=LBX PTR+1
LBX TOT=LBX TOT+1
LINK(ANDRON(NEST)), ANDCOL(NEST))=
* OR8(LINK(ANDRON(NEST)), ANDCOL(NEST))=char(33))
  LINK(ANDRON(NEST)), ANDCOL(NEST)-1)=
* OR8(LINK(ANDRON(NEST)), ANDCOL(NEST)-1), char(2))
  LINK(ANDRON(NEST)), ANDCOL(NEST)+3)=char(16)
  ANDCOL(NEST)=12
C -------—----- OUT -------—-----
elseif (FIELD1(CNTR) .eq. 'J') then
if (ANDCOL(NEST) .le. 12) then
do 45 J=ANDCOL(NEST),11
C {horizontal element)
  ELMNTC(INDEX,1)=char(1)
  ELMNTC(INDEX,2)=ANDRON(NEST)
  ELMNTC(INDEX,3)=J
  LINK(ANDRON(NEST),J)=OR8(LINK(ANDRON(NEST),J), char(33))
  LINK(ANDRON(NEST),J-1)=
* OR8(LINK(ANDRON(NEST),J-1),char(2))
  LINK(ANDRON(NEST),J+1)=char(16)
  INDEX=INDEX+1
45 con tinue
  ANDCOL(NEST)=12
endif
if (FIELD2(CNTR) .eq. 'B') then
  ELMNTC(INDEX,1)=char(10)
  TYPREF(FIELD3(CNTR))='Y'
e lseif (FIELD2(CNTR) .eq. 'C') then
  ELMNTC(INDEX,1)=char(11)
e lseif (FIELD2(CNTR) .eq. 'E') then
  ELMNTC(INDEX,1)=char(15)
  TYPREF(FIELD3(CNTR))='Y'
e lseif (FIELD2(CNTR) .eq. 'F') then
  ELMNTC(INDEX,1)=char(14)
e ndif
E L M NT C(INDEX,2)=ANDRON(NEST)
ELMNTC(INDEX,3)=12
ELMNTC(INDEX,3)=FIELD3(CNTR)
if (ANDRON(NEST) .eq. 1) then
  LINK(1,12)=char(48)
  LINK(1,11)=OR8(LINK(1,11),char(2))
  MARK=INDEX
else
  LINK(ANDRON(NEST),12)=char(96)
  LINK(ANDRON(NEST)-1,12)=OR8(LINK(ANDRON(NEST)-1,12),char(128))
endif
  ANDRON(NEST)=ANDRON(NEST)+1
else
  call BEEP(1,1)
  write(*,*),'CONVERSION ERROR - UNKNOWN ELEMENT'
  stop
endif
goto 100
end
APPENDIX G, ASSEMBLY LANGUAGE UTILITY ROUTINES

NOTE: The contents of this appendix were taken from a file containing assembly language source code for the utility routines. With the exception of this note and the headings on each page, all lines preceded by semicolons are comments and all remaining lines are executable code.

<table>
<thead>
<tr>
<th>Assembly subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHECKC, CHECKR</td>
<td></td>
</tr>
<tr>
<td>SETMOD</td>
<td>sets video mode and clears screen</td>
</tr>
<tr>
<td>CSROFF</td>
<td>turns cursor off</td>
</tr>
<tr>
<td>CSON</td>
<td>turns cursor on</td>
</tr>
<tr>
<td>LOCATE</td>
<td>moves cursor to the desired row/column on the specified page</td>
</tr>
<tr>
<td>SETPAG</td>
<td>sets the active page, for color text mode</td>
</tr>
<tr>
<td>UPSCRL</td>
<td>scrolls a text window up on the display screen</td>
</tr>
<tr>
<td>DNSCRL</td>
<td>scrolls a text window down on the display screen</td>
</tr>
<tr>
<td>GETPARM</td>
<td>assembly subroutine for the UPSCRL and DNSCRL routines</td>
</tr>
<tr>
<td>CLS</td>
<td>clears the entire screen, sets the attribute for the entire screen, and sets the border color to the background color</td>
</tr>
<tr>
<td>RDCHAT</td>
<td>returns the character and attribute at the current cursor position on the specified page</td>
</tr>
<tr>
<td>WRCHAT</td>
<td>writes a character and attribute at the current cursor position on the specified page and advances the cursor</td>
</tr>
<tr>
<td>WRCHAR</td>
<td>writes a character only at the current cursor position on the specified page and advances the cursor</td>
</tr>
<tr>
<td>WRTTY</td>
<td>writes a character at the current cursor position on the current page with the current attribute &amp; advances cursor</td>
</tr>
<tr>
<td>ERASLN</td>
<td>blanks a line on a specified page and row by writing 80 spaces using the specified attribute</td>
</tr>
<tr>
<td>BORDER</td>
<td>selects the border color (text mode only) or selects between the two color palettes (graphics mode only)</td>
</tr>
<tr>
<td>INKEY</td>
<td>removes next character from the keyboard buffer; returns its ASCII value and scan code or zero and the extended key code</td>
</tr>
<tr>
<td>IFKEY</td>
<td>returns 0 if keyboard buffer is empty; returns same data as INKEY if keyboard buffer is not empty</td>
</tr>
<tr>
<td>KYSHFT</td>
<td>returns the first keyboard shift byte</td>
</tr>
<tr>
<td>CLRBUF</td>
<td>clears the system's keyboard buffer</td>
</tr>
<tr>
<td>ASCBIN</td>
<td>converts five numeric ASCII characters to a 16 bit word</td>
</tr>
<tr>
<td>BINASC</td>
<td>converts a 16 bit word to five numeric ASCII characters</td>
</tr>
<tr>
<td>AND16</td>
<td>performs a logical AND on the two (2 byte) input parameters</td>
</tr>
<tr>
<td>OR16</td>
<td>performs a logical OR on the two (2 byte) input parameters</td>
</tr>
<tr>
<td>XOR16</td>
<td>performs a logical OR on the two (2 byte) input parameters</td>
</tr>
<tr>
<td>AND8</td>
<td>performs a logical AND on the two (1 byte) input parameters</td>
</tr>
</tbody>
</table>
5 OR8 — performs a logical OR on the two (2 byte) input parameters
5 XOR8 — performs a logical OR on the two (2 byte) input parameters
utilitycode segment 'code'

assume CS:utilitycode

public SETMOD
public CSROFF
public CSRON
public LOCATE
public SETPAG
public UPSCTRL
public DMSCTRL
public CLS
public RDCHAT
public WRCHAT
public WRCCHAR
public WRTTY
public ERASLN
public BORDER
public INKEY
public IFKEY
public KYSHFT
public CLRBUF
public ASCBIN
public BINASC
public AND16
public OR16
public XOR16
public AND8
public OR8
public XOR8
SUBROUTINES CHECKC AND CHECKR

; CHECKC - checks that the COLUMN parameter is in the range 1 to 80

CHECKC proc near ; check that parameter in BL is within limits
        cmp ah,80 ; compare param with upper limit (80)
        jbe okmaxc ; if param <= limit, then jump
        mov ah,79 ; if param > limit, then set param at limit-1
        ret ; return to caller
OKMAXC:  cmp ah,0 ; compare param with lower limit (0)
        je minc ; if param = limit, then jump
        dec ah ; decrement param by 1
        ret ; return to caller
MINC:    ret ; return to caller
CHECKC endp

; CHECKR - checks that the ROM parameter is in the range 1 to 25

CHECKR proc near ; check that parameter in BL is within limits
        cmp ah,25 ; compare param with upper limit (25)
        jbe okmaxr ; if param <= limit, then jump
        mov ah,24 ; if param > limit, then set param at limit-1
        ret ; return to caller
OKMAXR:  cmp ah,0 ; compare param with lower limit (0)
        je minr ; if param = limit, then jump
        dec ah ; decrement param by 1
        ret ; return to caller
MINR:    ret ; return to caller
CHECKR endp
SUBROUTINE SETMOD

; SETMOD - sets video mode and clears screen

; FORTRAN INTERFACE:
; INTEGER*2 MODE
; call SETMOD(MODE)

; NOTES: MODE = {0,1,2,3,4,5,6,7}

; MODE MEANING
; 0 40 x 25 BW
; 1 40 x 25 COLOR TEXT MODES
; 2 80 x 25 BW
; 3 80 x 25 COLOR
; 4 320 x 200 COLOR GRAPHICS MODES
; 5 320 x 200 BW
; 6 640 x 200 BW
; 7 80 x 25 BW (monochrome board only)

SETMOD proc far
; (uses INTERRUPT 16-0)

push ax ; save ax
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer

mov es,[bp]+14 ; get address segment of parameter (MODE)
mov si,[bp]+12 ; get address offset of parameter
mov al,es:[si] ; get parameter value
and al,7 ; limit MODE to a maximum of 7

mov ah,00 ; service code
int 16 ; interrupt - request service

pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop ax ; restore ax
ret 04 ; return to caller

SETMOD endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES 252
SUBROUTINES CSROFF AND CSRON

CSROFF - turns cursor off (makes it invisible)

FORTRAN INTERFACE:
call CSROFF

Note: This routine does not change the current cursor size.

CSROFF proc far
    push ax ; save ax
    push bx ; save bx
    push cx ; save cx
    push dx ; save dx
    mov ah, 15 ; service code to read current page
    int 16 ; interrupt - request service
    mov ah, 03 ; service code to read current cursor size
    int 16 ; interrupt - request service
    or ch, 20h ; turn bit 5 on to turn cursor off
    mov ah, 01 ; service code to set cursor type
    int 16 ; interrupt - request service
    pop dx ; restore dx
    pop cx ; restore cx
    pop bx ; restore bx
    pop ax ; restore ax
    ret ; return to caller

CSROFF endp

CSRON - turns cursor on (makes it visible)

FORTRAN INTERFACE:
call CSRON

Note: This routine does not change the current cursor size.

CSRON proc far
    push ax ; save ax
    push bx ; save bx
    push cx ; save cx
    push dx ; save dx
    mov ah, 15 ; service code to read current page
    int 16 ; interrupt - request service
    mov ah, 03 ; service code to read current cursor size
    int 16 ; interrupt - request service
    and ch, 0Fh ; turn bit 5 off to turn cursor on
    mov ah, 01 ; service code to set cursor type
    int 16 ; interrupt - request service
    pop dx ; restore dx
    pop cx ; restore cx
    pop bx ; restore bx
    pop ax ; restore ax
    ret ; return to caller

CSRON endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES 253
SUBROUTINE LOCATE

; LOCATE - moves the cursor to the desired row/column on a specified page

; FORTRAN INTERFACE:
; INTEGER*2 PAGE, ROW, COLUMN
; call LOCATE(PAGE, ROW, COLUMN)

; NOTES: PAGE = {0, 1, 2, 3}
; ROW = {1, 2, 3, ..., 25}
; COLUMN = {1, 2, 3, ..., 80}

LOCATE proc far
; (uses INTERRUPT 16 02)

push ax ; save ax
push bx ; save bx
push dx ; save dx
push es ; save es
push si ; save si
push bp ; save bp
mov es, [bp+26] ; get address segment of 1st parameter (PAGE)
mov si, [bp+24] ; get address offset of 1st parameter
mov bh, es:[si] ; get parameter value
and bh, 3 ; limit PAGE to a maximum of 3

mov es, [bp+22] ; get address segment of 2nd parameter (ROW)
mov si, [bp+20] ; get address offset of 2nd parameter
mov dh, es:[si] ; get parameter value
mov ah, dh ; pass parameter to CHECKR
call checkr ; check that parameter is within limits
mov dh, ah ; return parameter to DH

mov es, [bp+18] ; get address segment of 3rd param (COLUMN)
mov si, [bp+16] ; get address offset of 3rd parameter
mov dl, es:[si] ; get parameter value
mov ah, dl ; pass parameter to CHECKC
call checkc ; check that parameter is within limits
mov dl, ah ; return parameter to DL

mov ah, 02 ; service code
int 16 ; interrupt - request service

pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop dx ; restore dx
pop bx ; restore bx
pop ax ; restore ax
ret 12 ; return to caller

LOCATE endp
```
SUBROUTINE SETPAG

; SETPAG - sets the active page (color text mode only; cannot be used in color graphics mode or with monochrome display)

; FORTRAN INTERFACE:
; INTEGER*2 PAGE
; CALL SETPAG(PAGE)

; Notes: the page only applies in color text mode
; PAGE = (0,1,2,3)

SETPAG proc far ; (uses INTERRUPT 16 05)
    push ax ; save ax
    push es ; save es
    push si ; save si
    push bp ; save bp
    mov bp,sp ; get stack (parameter) pointer
    mov es,[bp+14] ; get address segment of parameter (PAGE)
    mov si,[bp+12] ; get address offset of parameter
    mov al,es:[si] ; get parameter value
    and al,3 ; limit PAGE to a maximum of 3
    mov ah,05 ; service code
    int 16 ; interrupt - request service
    pop bp ; restore bp
    pop si ; restore si
    pop es ; restore es
    pop ax ; restore ax
    ret 04 ; return to caller

SETPAG endp
```

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES
SUBROUTINE UPSCRL

UPSCRL - scrolls a text window up on display screen, setting the desired attribute on the new lines

FORTRAN INTERFACE:
INTEGER*2 LINES, TOP, BOTTOM, LEFT, RIGHT, ATTRIB
CALL UPSCRL(LINES, TOP, BOTTOM, LEFT, RIGHT, ATTRIB)

Notes: TOP/BOTTOM/LEFT/RIGHT specify the boundaries of the window;
LINES is the number of lines to scroll; ATTRIB sets the display attribute for the blank lines scrolled in;
LINES = 0 causes blanking of entire window
LINES | TOP | BOTTOM = {1,2,3, ..., 25}
LEFT | RIGHT = {1,2,3, ..., 80}

UPSCRL proc far ; (uses INTERRUPT 16 06)
push ax ; save ax
push bx ; save bx
push cx ; save cx
push dx ; save dx
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
call getparm ; get parameters from stack
mov ah,06 ; service code
int 16 ; interrupt - request service
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop dx ; restore dx
pop cx ; restore cx
pop bx ; restore bx
pop ax ; restore ax
ret 24 ; return to caller

UPSCRL endp
SUBROUTINE DNSCRL

; DNSCRL - scrolls a text window down on display screen, setting the desired
; attribute on the new lines

; FORTRAN INTERFACE:
; INTEGER*2 LINES, TOP, BOTTOM, LEFT, RIGHT, ATTRIB
; call DNSCRL(LINES, TOP, BOTTOM, LEFT, RIGHT, ATTRIB)

; Notes: TOP/BOTTOM/LEFT/RIGHT specify the boundaries of the window;
; LINES is the number of lines to scroll; ATTRIB sets the
; display attribute for the blank lines scrolled in;
; LINES = 0 causes blanking of entire window
; LINES | TOP | BOTTOM = (1, 2, 3, ..., 25)
; LEFT | RIGHT = (1, 2, 3, ..., 80)

DNSCRL proc far ; (uses INTERRUPT 16 07)
push ax ; save ax
push bx ; save bx
push cx ; save cx
push dx ; save dx
push es ; save es
push si ; save si
push bp ; save bp
mov bp, sp ; get stack (parameter) pointer
call getparm ; get parameters from stack
mov ah, 07 ; service code
int 16 ; interrupt - request service
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop dx ; restore dx
pop cx ; restore cx
pop bx ; restore bx
pop ax ; restore ax
ret 24 ; return to caller

DNSCRL endp
SUBROUTINE GETPARM

; GETPARM - gets parameters from stack for the UPSCRL and DNSCRL subroutines

GETPARM proc near

; get parameters from stack

mov es,[bp+40] ; get address segment of 1st param (LINES)
mov si,[bp+38] ; get address offset of 1st parameter
mov al,es:[si] ; get parameter value
cmp al,0 ; check that parameter is within limits
jae oklines ; if LINES => 0, then jump
mov al,1 ; set LINES to 1 if it is below 0

OKLINES:
mov es,[bp+36] ; get address segment of 2nd param (TOP-1)
mov si,[bp+54] ; get address offset of 2nd parameter
mov ch,es:[si] ; get parameter value
mov ah,ch ; pass parameter to CHECKR
call checkr ; check that parameter is within limits
mov ch,ah ; return parameter to CH

mov es,[bp+32] ; get address segment of 3rd param (BOTTOM-1)
mov si,[bp+20] ; get address offset of 3rd parameter
mov dh,es:[si] ; get parameter value
mov ah,dh ; pass parameter to CHECKR
call checkr ; check that parameter is within limits
mov dh,ah ; return parameter to DH

mov es,[bp+24] ; get address segment of 4th param (LEFT-1)
mov si,[bp+26] ; get address offset of 4th parameter
mov cl,es:[si] ; get parameter value
mov ah,cl ; pass parameter to CHECKR
call checkc ; check that parameter is within limits
mov cl,ah ; return parameter to CL

mov es,[bp+22] ; get address segment of 5th param (RIGHT-1)
mov si,[bp+28] ; get address offset of 5th parameter
mov dl,es:[si] ; get parameter value
mov ah,dl ; pass parameter to CHECKR
call checkc ; check that parameter is within limits
mov dl,ah ; return parameter to DL

mov es,[bp+20] ; get address segment of 6th param (ATTRIB)
mov si,[bp+18] ; get address offset of 6th parameter
mov bh,es:[si] ; get parameter value

ret

GETPARM endp
SUBROUTINE CLS

; CLS - clears the entire screen, sets the attribute for the entire
; screen, and sets the border color to the background color

; FORTRAN INTERFACE:
; INTEGER*2 ATTRIB
; call CLS(ATTRIB)

CLS proc far

; (uses INTERRUPT 16 06 & INTERRUPT 16 11)

    push ax ; save ax
    push bx ; save bx
    push cx ; save cx
    push dx ; save dx
    push es ; save es
    push si ; save si
    push bp ; save bp

    mov bp,sp ; get stack (parameter) pointer

    mov al,0 ; set parameter value (LINES)
    mov ch,0 ; set parameter value (TOP)
    mov dh,24 ; set parameter value (BOTTOM)
    mov cl,0 ; set parameter value (LEFT)
    mov dl,79 ; set parameter value (RIGHT)

    mov es,[bp+20] ; get address segment of parameter (ATTRIB)
    mov si,[bp+18] ; get address offset of parameter
    mov bh,es:[si] ; get parameter value

    mov ah,06 ; service code
    int 16 ; interrupt - request service

    mov bl,112 ; set background bits in the attribute byte
    and bl,bh ; isolate background (bg) bits in BH

    mov cl,4 ; shift 4 times
    shr bl,cl ; shift bg bits to foreground bits position

    mov bh,0 ; set color ID

    mov ah,11 ; service code
    int 16 ; interrupt - request service

    pop bp ; restore bp
    pop si ; restore si
    pop es ; restore es
    pop dx ; restore dx
    pop cx ; restore cx
    pop bx ; restore bx
    pop ax ; restore ax

    ret 04 ; return to caller

CLS endp
SUBROUTINE RDCHAT

; RDCHAT - returns the character and attribute at the current cursor position on the specified page

; FORTRAN INTERFACE:
; INTEGER*2 PAGE, ATTRIB
; CHARACTER*1 CHAR
; call RDCHAT(PAGE,CHAR,ATTRIB)

RDCHAT proc far  ; (uses INTERRUPT 16 08)
    push ax   ; save ax
    push bx   ; save bx
    push es   ; save es
    push si   ; save si
    push bp   ; save bp
    mov bp,sp ; get stack (parameter) pointer
    mov es,[bp+24] ; get address segment of 1st parameter (PAGE)
    mov si,[bp+22] ; get address offset of 1st parameter
    mov bh,es:[si] ; get parameter value
    and bh,3 ; limit PAGE to a maximum of 3
    push bp ; save parameter pointer
    mov ah,08 ; service code
    int 16 ; interrupt - request service
    pop bp ; restore parameter pointer
    mov es,[bp+20] ; get address segment of 2nd parameter (CHAR)
    mov si,[bp+18] ; get address offset of 2nd parameter
    mov es:[si],al ; pass back value
    mov es,[bp+16] ; get address segment of 3rd param (ATTRIB)
    mov si,[bp+14] ; get address offset of 3rd parameter
    mov es:[si],ah ; pass back value
    inc si ; set address of next higher byte
    mov byte ptr es:[si],0 ; zero next higher byte
    pop bp ; restore bp
    pop si ; restore si
    pop es ; restore es
    pop bx ; restore bx
    pop ax ; restore ax
    ret 12 ; return to caller

RDCHAT endp
SUBROUTINE WRCHAT

; WRCHAT - writes a character and attribute at the current cursor position
; on the specified page and advances the cursor; multiple copies
; of the character can be written (e.g. for blanking)

; FORTRAN INTERFACE:
; INTEGER*2 PAGE, ATTRIB, COUNT
; CHARACTER*1 CHAR
; call WRCHAT(PAGE,CHAR,ATTRIB,COUNT)

WRCHAT proc far ; (uses INTERRUPT 16 9)
push ax ; save ax
push bx ; save bx
push cx ; save cx
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
mov es,[bp+30] ; get address segment of 1st parameter (PAGE)
mov si,[bp+28] ; get address offset of 1st parameter
mov bh,es:[si] ; get parameter value
and bh,3 ; limit PAGE to a maximum of 3
mov es,[bp+26] ; get address segment of 2nd parameter (CHAR)
mov si,[bp+24] ; get address offset of 2nd parameter
mov al,es:[si] ; get parameter value
mov es,[bp+22] ; get address segment of 3rd param (ATTRIB)
mov si,[bp+20] ; get address offset of 3rd parameter
mov bl,es:[si] ; get parameter value
mov es,[bp+18] ; get address segment of 4th param (COUNT)
mov si,[bp+16] ; get address offset of 4th parameter
mov cx,es:[si] ; get parameter value
mov ah,09 ; service code to write CHAR and ATTRIB
int 16 ; interrupt - request service
mov ah,3 ; service code to read cursor position
int 16
inc dl ; increment cursor column
mov ah,2 ; service code to set cursor position
int 16
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop cx ; restore cx
pop bx ; restore bx
pop ax ; restore ax
ret 16 ; return to caller

WRCHAT endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES
**SUBROUTINE WRCHAR**

; WRCHAR - writes a character only at the current cursor position on the
; specified page and advances the cursor. Multiple copies of the
; character can be written (e.g. for blanking)

; FORTRAN INTERFACE:
; INTEGER*2 PAGE, COUNT
; CHARACTER*1 CHAR
; call WRCHAR(PAGE,CHAR,COUNT)

WRCHAR proc far ; (uses INTERRUPT 16 10)
push ax ; save ax
push bx ; save bx
push cx ; save cx
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
mov es,[bp+26] ; get address segment of 1st parameter (PAGE)
mov si,[bp+24] ; get address offset of 1st parameter
mov bh,es:[si] ; get parameter value
and bh,3 ; limit PAGE to a maximum of 3
mov es,[bp+22] ; get address segment of 2nd parameter (CHAR)
mov si,[bp+20] ; get address offset of 2nd parameter
mov al,es:[si] ; get parameter value
mov es,[bp+18] ; get address segment of 3rd param (COUNT)
mov si,[bp+16] ; get address offset of 3rd parameter
mov cx,es:[si] ; get parameter value
mov ah,10 ; service code to write CHAR
int 16 ; interrupt - request service
mov ah,3 ; service code to read cursor position
int 16
inc dl ; increment cursor column
mov ah,2 ; service code to set cursor position
int 16
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop cx ; restore cx
pop bx ; restore bx
pop ax ; restore ax
ret 12 ; return to caller

WRCHAR endp
SUBROUTINE WRTTY

; WRTTY - writes a character at the current cursor position on the current page with the current attribute and advances the cursor

; FORTRAN INTERFACE:
; CHARACTER*1 CHAR
; call WRTTY(CHAR)

WRTTY proc far ; (uses INTERRUPT 16 14)

push ax ; save ax
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
mov es,[bp+14] ; get address segment of 2nd parameter (CHAR)
mov si,[bp+12] ; get address offset of 2nd parameter
mov al,es:[si] ; get parameter value
mov ah,14 ; service code
int 16 ; interrupt - request service
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop ax ; restore ax
ret 4 ; return to caller

WRTTY endp
SUBROUTINE ERASLN

; ERASLN - blanks a line on a specified page and row by writing 80 spaces
; using the specified attribute

; FORTRAN INTERFACE:
; INTEGER*2 PAGE, ROW, ATTRIB
; call ERASLN(PAGE,ROW,ATTRIB)

ERASLN proc far ; (uses INTERRUPT 16 10)
push ax ; save ax
push bx ; save bx
push cx ; save cx
push dx ; save dx
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
push es ; get address segment of 1st parameter (PAGE)
push si ; get address offset of 1st parameter
mov bh,es:[si] ; get parameter value
and bh,3 ; limit PAGE to a maximum of 3
push es ; get address segment of 2nd parameter (ROW)
push si ; get address offset of 2nd parameter
mov dh,es:[si] ; get parameter value
mov ah,dh ; pass parameter to CHECKR
call checkr ; check that parameter is within limits
mov dh,ah ; return parameter to DH
push es ; get address segment of 3rd param (ATTRIB)
push si ; get address offset of 3rd parameter
mov bl,es:[si] ; get parameter value
mov dl,0 ; set column parameter at column 1
mov ah,2 ; service code to set cursor position
int 16 ; interrupt - request service
mov al,32 ; set 'space' character
mov cx,80 ; write 80 characters
int 16 ; interrupt - request service
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop dx ; restore dx
pop cx ; restore cx
pop bx ; restore bx
pop ax ; restore ax
ret 08 ; return to caller

ERASLN endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES
SUBROUTINE BORDER

; BORDER - selects the border color (text mode only) or
; selects between the two color palettes (graphics mode only)

; FORTRAN INTERFACE:
; INTEGER*2 COLOR
; call BORDER(COLOR)

; NOTE: COLOR = {0,1,2,...15}
; LOW INT HIGH INT
; BLACK 0 8
; BLUE 1 9
; GREEN 2 10
; CYAN 3 11
; RED 4 12
; MAGENTA 5 13
; BROWN 6 14
; WHITE 7 15

BORDER proc far  ; (uses INTERRUPT 16 11)
push ax ; save ax
push bx ; save bx
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
mov es,[bp]+16 ; get address segment of parameter (COLOR)
mov si,[bp]+14 ; get address offset of parameter
mov bl,es:[si] ; get parameter value
and bl,15 ; limit COLOR to maximum of 15
mov ah,011 ; service code
int 16 ; interrupt - request service
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop bx ; restore bx
pop ax ; restore ax
ret 04 ; return to caller

BORDER endp
SUBROUTINE INKEY

; INKEY - removes next character from the keyboard buffer; returns its
; ASCII value and scan code or zero and the extended key code

; FORTRAN INTERFACE:
; INTEGER*2 KYCOD1, KYCOD2
; call INKEY(KYCOD1,KYCOD2)

; Notes: KYCOD1 is the ASCII value of the character and KYCOD2 is the scan
; code of its key; for extended code keys, KYCOD1 is zero and KYCOD2
; is the extended code; if keyboard buffer is empty, program operation
; halts until a key is depressed

INKEY proc far ; (uses INTERRUPT 22 00)
push ax ; save ax
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
push bp ; save parameter pointer
mov ah,00 ; service code
int 22 ; interrupt - request service
pop bp ; restore parameter pointer
mov es,[bp+18] ; get address segment of parameter (KYCOD1)
mov si,[bp+16] ; get address offset of parameter
inc si ; set address of next higher byte
mov byte ptr es:[si+0],0 ; zero next higher byte
mov es,[bp+14] ; get address segment of parameter (KYCOD2)
mov si,[bp+12] ; get address offset of parameter
mov es:[si+ah],al ; pass back value
inc si ; set address of next higher byte
mov byte ptr es:[si+0],0 ; zero next higher byte
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop ax ; restore ax
ret 08 ; return to caller

INKEY endp
SUBROUTINE IFKEY

IFKEY - does not remove next character from the keyboard buffer;
returns 0 if keyboard buffer is empty; returns same data as
INKEY if keyboard buffer is not empty.

FORTRAN INTERFACE:
INTEGER*2 KYCOD1, KYCOD2
call IFKEY(KYCOD1,KYCOD2)

Notes: if no keystroke is available, the returned values are zero;
otherwise the returned data is same as for INKEY;
Keystroke data is not removed from the keyboard buffer.

IFKEY proc far
; (uses INTERRUPT 22 01)
push ax  ; save ax
push es  ; save es
push si  ; save si
push bp  ; save bp
mov bp,sp ; get stack (parameter) pointer
push bp  ; save parameter pointer
mov ah,01 ; service code
int 22 ; interrupt - request service
pop bp  ; restore parameter pointer
jnz jump_1 ; if no code available
mov ax,00 ; then set return values to zero
jump_1:
mov es,[bp+18] ; get address segment of parameter (KYCOD1)
mov si,[bp+16] ; get address offset of parameter
mov es:[si],al ; pass back value
inc si ; set address of next higher byte
mov byte ptr es:[si],0 ; zero next higher byte
mov es,[bp+14] ; get address segment of parameter (KYCOD2)
mov si,[bp+12] ; get address offset of parameter
mov es:[si],ah ; pass back value
inc si ; set address of next higher byte
mov byte ptr es:[si],0 ; zero next higher byte
pop bp  ; restore bp
pop si  ; restore si
pop es  ; restore es
pop ax  ; restore ax
ret 08 ; return to caller

IFKEY endp
SUBROUTINE KYSHFT

KYSHFT - returns the first keyboard shift byte

FORTRAN INTERFACE:
CHARACTER*1 STATE
CALL KYSHFT(STATE)

Note: BIT MEANING BIT MEANING
0 insert state 4 alt key state
1 caps lock state 5 ctrl key state
2 num lock state 6 left shift key state
3 scroll lock state 7 right shift key state

bit = 1 means state is on or key is depressed
bit = 0 means state is off or key is not depressed

KYSHFT proc far ; uses INTERRUPT 22 02
push ax ; save ax
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
push bp ; save parameter pointer
mov ah,02 ; service code
int 22 ; interrupt - request service
pop bp ; restore parameter pointer
mov es,[bp+14] ; get address segment of parameter (STATE)
mov si,[bp+12] ; get address offset of parameter
mov es:[si],al ; pass back value
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop ax ; restore ax
ret 04 ; return to caller

KYSHFT endp
SUBROUTINE CLRBUF

CLRBUF - clears the system's keyboard buffer

FORTRAN INTERFACE:

call CLRBUF

NOTE: This routine clears the keyboard buffer (a ring buffer) by setting
the pointer which indicates the head of the buffer (BUFFER_HEAD)
equal to the pointer which indicates the tail of the buffer
(BUFFER_TAIL). More information on how the keyboard buffer works
is located on pages 5-31 (ROM BIOS Data Area) and 5-54 through 5-64
(Keyboard I/O) of the IBM PC Hardware Technical Reference Manual.

CLRBUF proc far

push ds ; save DS
push ax ; save AX
push bx ; save BX

mov ax, 40h ; set ROM BIOS data segment
mov ds, ax
mov bx, 1AH ; set offset of BUFFER_HEAD
mov ax, [bx + 2] ; get BUFFER_TAIL address
mov [bx], ax ; set BUFFER_HEAD equal to BUFFER_TAIL

pop bx ; restore BX
pop ax ; restore AX
pop ds ; restore DS
ret

CLRBUF endp
SUBROUTINE ASCBIN

; ASCBIN - convert five numeric ASCII characters to a 16 bit word (integer*2)
; and return it in parameter VALUE

; FORTRAN INTERFACE:
; character*1 CHAR(5)
; integer*2 VALUE
; call ASCBIN(CHAR(1),CHAR(2),CHAR(3),CHAR(4),CHAR(5),VALUE)

; Notes:  CHAR = ('0','1','2','3','4','5','6','7','8','9')
; VALUE = (0..32767)
; The integer represented by the ASCII character string is limited
; to the range 0..32767. Negative characters are not valid. If an
; input character is non-numeric, then DECBIN assumes it is 0.
; (i.e. leading spaces are treated as 0)

ASCBIN proc far
push ax        ; save ax
push bx        ; save bx
push cx        ; save cx
push dx        ; save dx
push es        ; save es
push si        ; save si
push bp        ; save bp
mov bp,sp      ; get stack (parameter) pointer
mov bx,10      ; initialize BX to 10 (decimal multiplier)
mov cx,5       ; initialize CX to 5 (number of characters)
xor dx,dx      ; initialize DX to 0 (initial integer value)

BEGIN:         mov ax,4      ; multiplier of 4 (4 bytes per address)
mul cl         ; multiply CL by AL (CL by 4) (product in AX)
mov si,ax      ; place product in SI
mov es,[bp+si+20] ; get address segment of param (CHAR(i))
mov si,[bp+si+18] ; get address offset of parameter
mov al,es:[si] ; get parameter value
sub al,48      ; check if input char is < ASCII '0'
jl alpha       ; zero AL if below
cmp al,9       ; check if input char is <= ASCII '9'
jbe convert   ; convert to value if below or equal to

ALPHA:         xor al,al   ; zero AL (non-numeric input character)

CONVERT:       cbw         ; convert AL to word AX (place 0 in AH)
push ax        ; save digit
mul bx         ; multiply AX by BX (AX by 10)
; (low order word of product is in AX)
; (high order word of product is in DX)
mov dx,ax      ; place product in DX
pop ax         ; restore digit
add dx,ax      ; add digit to accumulated value
loop begin     ; get next character if CX > 0
mov es,[bp+20] ; get address segment of parameter (VALUE)
mov si,[bp+18] ; get address offset of parameter
mov es:[si],dx ; pass back value
pop bp         ; restore bp
pop si         ; restore si
pop es         ; restore es
pop dx         ; restore dx
pop cx         ; restore cx
pop bx         ; restore bx
pop ax         ; restore ax
ret 24         ; return to caller

ASCBIN endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES 270
SUBROUTINE BINASC

; BINASC - convert a 16 bit word (integer*2) to five numeric ASCII characters

; Notes: The input value must be greater than 0 and less than 32768; the
; value is not checked. Although an integer*2 value is a signed
; integer, it is treated as an unsigned integer in this routine.

; FORTRAN INTERFACE:
; integer*2 VALUE
; character*1 CHAR(5)
; call BINASC(VALUE,CHAR(1),CHAR(2),CHAR(3),CHAR(4),CHAR(5))

; Notes: VALUE = (0..32767)
; CHAR = ('0','1','2','3','4','5','6','7','8','9')
; VALUE must be a positive integer.

BINASC proc far

push ax ; save ax
push bx ; save bx
push cx ; save cx
push dx ; save dx
push es ; save es
push si ; save si
push bp ; save bp

mov bp,sp ; get stack (parameter) pointer
mov es,[bp+40] ; get address segment of parameter (VALUE)
mov si,[bp+58] ; get address offset of parameter
mov dx,es:[si] ; get value to be converted (dividend)

xor cx,cx ; initialize the counter to 0
mov bx,10 ; initialize BX to 10 (decimal divisor)

loop1:
mov ax,dx ; put value into low word of dividend (DX:AX)
xor dx,dx ; zero high word of dividend
div bx ; divide DX:AX by BX (dividend by 10)
; (quotient is in AX & remainder is in DX)
xchg ax,dx ; put quotient in DX & remainder in AX
add al,48 ; change remainder digit to ASCII character
inc cx ; count the character

push ax ; save the character
mov al,4 ; multiplier of 4 (4 bytes per address)
mul cl ; multiply CL by AL (CL by 4) (product in AX)
mov si,ax ; place product in SI
pop ax ; restore the character
mov es,[bp+si+16] ; get address segment of param (CHAR(i))
mov si,[bp+si+14] ; get address offset of parameter
mov es:[si],al ; pass back character

cmp dx,0 ; done?
ja loop1 ; repeat if quotient is above 0

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES 271
pass back null character to remaining character parameters

```assembly
mov bx,cx ; set BX to number of characters
mov cx,5 ; initialize counter to 5 character positions

LOOP2:
    cmp cx,bx ; no. of positions left = no. of characters?
    je exit ; exit if true
    mov al,4 ; multiplier of 4 (4 bytes per address)
    mul cl ; multiply CL by AL (CL by 4) (product in AX)
    mov si,ax ; place product in SI
    mov es,[bp+si+16] ; get address segment of param (CHAR(i))
    mov si,[bp+si+14] ; get address offset of parameter
    mov byte ptr es:[si],0 ; set parameter to null character
    loop loop2 ; repeat if CX > 0

EXIT:
    pop bp ; restore bp
    pop si ; restore si
    pop es ; restore es
    pop dx ; restore dx
    pop cx ; restore cx
    pop bx ; restore bx
    pop ax ; restore ax
    ret 24 ; return to caller
```

BINASC endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES
SUBROUTINE AND16

; AND16 - performs a logical AND on the two (2 byte) input parameters

; FORTRAN INTERFACE:
; INTEGER*2 AND16, PARM1, PARM2, X
; X = AND16(PARM1, PARM2)

; Notes: The result of this function is returned to the calling program
; in register AX.

AND16 proc far
push bx        ; save bx
push es        ; save es
push si        ; save si
push bp        ; save bp
mov bp,sp      ; get stack (parameter) pointer
mov es,[bp+18] ; get address segment of parameter (PARM1)
mov si,[bp+16] ; get address offset of parameter
mov ax,es:[si] ; get PARM1 value
mov es,[bp+14] ; get address segment of parameter (PARM2)
mov si,[bp+12] ; get address offset of parameter
mov bx,es:[si] ; get PARM2 value
and ax,bx      ; AND PARM1 & PARM2
pop bp         ; restore bp
pop si         ; restore si
pop es         ; restore es
pop bx         ; restore bx
ret 08         ; return to caller

AND16 endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES 273
SUBROUTINE OR16

; OR16 - performs a logical OR on the two (2 byte) input parameters

; FORTRAN INTERFACE:
; INTEGER*2 OR16, PARM1, PARM2, X
; X = OR16(PARM1,PARM2)

; Notes: The result of this function is returned to the calling program
; in register AX.

OR16 proc far

    push bx ; save bx
    push es ; save es
    push si ; save si
    push bp ; save bp
    mov bp,sp ; get stack (parameter) pointer
    mov es,[bp+18] ; get address segment of parameter (PARM1)
    mov si,[bp+16] ; get address offset of parameter
    mov ax,es:[si] ; get PARM1 value
    mov es,[bp+14] ; get address segment of parameter (PARM2)
    mov si,[bp+12] ; get address offset of parameter
    mov bx,es:[si] ; get PARM2 value
    or ax,bx ; OR PARM1 & PARM2
    pop bp ; restore bp
    pop si ; restore si
    pop es ; restore es
    pop bx ; restore bx
    ret 08 ; return to caller

OR16 endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES 274
SUBROUTINE XOR16

; XOR16 - performs a logical XOR on the two (2 byte) input parameters

; FORTRAN INTERFACE:
; INTEGER*2 XOR16, PARM1, PARM2, X
; X = XOR16(PARM1,PARM2)

; Notes: The result of this function is returned to the calling program
; in register AX.

XOR16 proc far

push bx ; save bx
push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer

mov es,[bp+18] ; get address segment of parameter (PARM1)
mov si,[bp+16] ; get address offset of parameter
mov ax,es:[si] ; get PARM1 value

mov es,[bp+14] ; get address segment of parameter (PARM2)
mov si,[bp+12] ; get address offset of parameter
mov bx,es:[si] ; get PARM2 value
xor ax,bx ; XOR PARM1 & PARM2

pop bp ; restore bp
pop si ; restore si
pop es ; restore es
pop bx ; restore bx
ret 08 ; return to caller

XOR16 endp

Appendix G. ASSEMBLY LANGUAGE UTILITY ROUTINES
SUBROUTINE AND8

; AND8 - performs a logical AND on the two (1 byte) input parameters

; FORTRAN INTERFACE:
; CHARACTER*1 PARM1, PARM2, X
; INTEGER*2 AND8
; X = AND8(PARM1,PARM2)

; Notes: The result of this function is returned to the calling program
; in register AX, although only the low byte in AL is used.

AND8 proc far

push es ; save es
push si ; save si
push bp ; save bp
mov bp,sp ; get stack (parameter) pointer
mov es,[bp+16] ; get address segment of parameter (PARM1)
mov si,[bp+14] ; get address offset of parameter
mov al,es:[si] ; get PARM1 value
mov es,[bp+12] ; get address segment of parameter (PARM2)
mov si,[bp+10] ; get address offset of parameter
mov ah,es:[si] ; get PARM2 value
and al,ah ; AND PARM1 & PARM2
xor ah,ah ; zero the high byte
pop bp ; restore bp
pop si ; restore si
pop es ; restore es
ret 08 ; return to caller

AND8 endp
SUBROUTINE OR8

; OR8 - performs a logical OR on the two (2 byte) input parameters

; FORTRAN INTERFACE:
; CHARACTER*1 PARM1, PARM2, X
; INTEGER*2 OR8
; X = OR8(PARM1, PARM2)

; Notes: The result of this function is returned to the calling program
; in register AX, although only the low byte in AL is used.

OR8 proc far

push es   ; save es
push si   ; save si
push bp   ; save bp
mov bp,sp ; get stack (parameter) pointer

mov es,[bp+16] ; get address segment of parameter (PARM1)
mov si,[bp+14] ; get address offset of parameter
mov al,es:[si] ; get PARM1 value

mov es,[bp+12] ; get address segment of parameter (PARM2)
mov si,[bp+10] ; get address offset of parameter
mov ah,es:[si] ; get PARM2 value

or al,ah ; OR PARM1 & PARM2
xor ah,ah ; zero the high byte

pop bp   ; restore bp
pop si   ; restore si
pop es   ; restore es
ret 08   ; return to caller

OR8 endp
SUBROUTINE XOR8

; XOR8 - performs a logical XOR on the two (2 byte) input parameters

; FORTRAN INTERFACE:
; CHARACTER*1 PARM1, PARM2, X
; INTEGER*2 XOR8
; X = XOR8(PARM1, PARM2)

; Notes: The result of this function is returned to the calling program
; in register AX, although only the low byte in AL is used.

XOR8 proc far

push es                ; save es
push si                ; save si
push bp                ; save bp
mov bp,sp              ; get stack (parameter) pointer
mov es,[bp+16]         ; get address segment of parameter (PARM1)
mov si,[bp+14]         ; get address offset of parameter
mov al,es:[si]         ; get PARM1 value
mov es,[bp+12]         ; get address segment of parameter (PARM2)
mov si,[bp+10]         ; get address offset of parameter
mov ah,es:[si]         ; get PARM2 value
xor al,ah              ; OR PARM1 & PARM2
xor ah,ah              ; zero the high byte
pop bp                 ; restore bp
pop si                 ; restore si
pop es                 ; restore es
ret                    ; return to caller

XOR8 endp

utilitycode ends
end
:athesis.