1987

A feasibility study of the development of an IBM PC-based programmable controller ladder logic programming system /

Steven M. Howard
Lehigh University

A Feasibility Study of the Development of an IBM PC-based Programmable Controller Ladder Logic Programming System

by

Steven M. Howard

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Manufacturing Systems Engineering

Lehigh University

1987

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 11, 1987
(date)

M B Moore
Professor in Charge

D. L Walker EE DIV. HEAD
for
Chairman of Department

Roger N Nagel
Director, MSE Program

Acknowledgments

First, I'd like to thank my family, my dad Bernie, my mom Carol, my sister Karen, and my brother Mark, for their understanding while I spent these long nine months hacking away at my computer instead of visiting them at home. Special thanks to my mom for the leftovers, especially the chicken soup and beef stew, that were my sustenance during many quick dinners.

Next, I'd like to thank Professor Groover for allowing me the freedom to construct this thesis to test my own ideas. I certainly appreciate his patient reading of my rough drafts and the many excellent comments that he offered for shaping this thesis into its final form.

Also, I'd like to thank my girlfriend Sue for sharing the joy of my accomplishments and having the patience to deal with my lousy moods when things just weren't working out.

Finally, I'd like to thank the Du Pont Company for employing me while I finished my thesis and allowing me some time to get it done.

Table of Contents

List of Figures

Abstract

Today, the programmable logic controller (PLC) is being used in an increasing number of applications for machine control and higher-level cell coordination. This trend is largely due to the decreasing cost/increasing power of microprocessor chip technology. The affordability of the new generation of PLCs justifies their growing use in industrial applications.

This trend has stimulated demand in two areas associated with PLC programming. The first is the need for new PLC programmers. The second area is the development of computer aided programming tools.

There are two drawbacks to the current documentation/off-line programming systems. First, documentation and off-line programming are sold as two separate packages that can operate on a common database of programs, but not as a totally integrated system. Separate packages allow the end user to purchase only the functionality that he requires. However, this marketing methodology forfeits advanced functions that could be implemented on an integrated system. The second drawback is that today's packages do not include a program simulation capability.

My thesis is to demonstrate the feasibility of developing an integrated programming, documentation, and

simulation package for developing ladder logic programs for programmable controllers. This software is for use on an IBM PC-compatible computer.

The purpose of this project is two-fold. Primarily, this software package is meant to serve as a teaching aid for ladder logic programming. In addition, it is the author's hope that this program will pioneer the research for a simple, yet powerful, ladder logic software development tool that would be useful in industrial applications.

1. Introduction to Programmable Controllers

1.1. Definition of a Programmable Controller

The National Electrical Manufacturers Association (NEMA) defines a programmable logic controller as a digitally operating electronic apparatus that uses a programmable memory for the internal storage of instructions for implementing specific functions such as logic, sequencing, timing, counting, and arithmetic, to control machines or processes through digital or analog input or output (I/O) modules [11]. Programmable logic controller is often shortened in the trade literature to programmable controller and abbreviated with the acronym PC. Use of the acronym PC for programmable controller leads to confusion with today's popular personal computers. In this thesis report, the acronym PLC will stand for the programmable logic controller while the acronym PC will be used exclusively to describe the personal computer; however, programmable controller will be used interchangeably to mean programmable logic controller.

The NEMA definition of programmable controllers is long-winded and confusing. Separating the definition into its components makes it easier to understand.

First, the PLC is a programmable, digital device. Although it is highly specialized, the PLC is still just a computer. Like any computer, the PLC's brain is the

central processing unit (CPU). In today's PLCs, the CPU is normally a microprocessor.

Second, the PLC has programmable memory, either static, EPROM, or volatile RAM, in which to store its program.

Third, the PLC has a number of functions, known as the instruction set, which can be combined to form the control program.

Finally, the PLC has input/output (I/O) devices which allow the PLC to communicate with the outside world. The I/O devices are connected to sensors (for example, photoelectric cells, limit switches, proximity switches, tachometers, and thermocouples) and actuators (for example, solenoid valves and motor starters) on the machine or process to be controlled. The input devices convert the feedback signals from the sensors into digital signals that the PLC can understand. Based on the digital signals read from the input devices, the PLC executes the control program to generate commands to the output devices. The output devices convert these commands into control signals to the actuators which perform the actual control of the machine or process.

1.2. History of the Programmable Controller

Now that a PLC has been defined, the next step is to understand why this specialized computer was needed. In 1966, General Motors developed a specification (for its Hydramatic Division) for a controller that was programmable. This controller would be used as a replacement for large relay control panels. GM wanted to reduce the tremendous effort and expense that was associated with the yearly changeover of car models. Up to this time, changeover of the relay panels that controlled the automated production lines was accomplished by rewirng the panels. In some of the larger panels, it was more cost effective to scrap the old panels and rebuild new ones from scratch rather than try to modify the hundreds of relays and thousands of wires in the old relay panels. GM engineers felt that a "programmable controller" would be a new, efficient method to meet the changing control requirements. No longer would the engineers deal with hardwired relays. Instead, they would work with "soft-wired" relays in a control program.

In 1969, Modicon (now Gould) introduced the first commercial PLC, the Modicon 084, to meet the requirements of the GM specification [7]. With the advent of this PLC, installation of control systems was fairly easy because the hard wiring associated with

relays was eliminated. Production line changeover time was reduced significantly because only the programming logic needed to be modified, a relatively simple procedure, to change the control of the lines. In addition, the programmability of the PLC meant that it was reuseable, which was a tremendous cost savings.

1.3. Programmable Controllers:

Simple, Time-proven Technology

Today, after nearly 20 years of exposure to harsh industrial environments, an environment atypical to most computers, the PLC has gained acceptance as the workhorse of the manufacturing plant. This acceptance has been earned because the PLC is a simple device that does its job, the control of machines and processes, and it does this job well. Today's PLCs are extremely reliable. These opinions about today's PLCs were summed up by Alex C. Mair, vice president, General Motors Corp. and president of the Engineering Society of Detroit, in his keynote address at the Programmable Controllers Conference in April 1986. He said, "Programmable controllers are old technology. They've been around for more than 20 years. The issue with programmable controllers isn't really new technology--it's good technology and programmable controllers are good technology. They're simple, versatile, and time-

proven. And, perhaps most important, they can function in a plant environment" [2].

Because of the dependability of the PLC, they are becoming more than the simple hardwired relay replacers as intended in the original GM specification. Today, PLCs are integrating large processes through distributed control networks. PLCs no longer are limited to the control of a single machine. Larger programmable controllers are capable of controlling multiple machines or processes. Communications networks enable larger PLCs to serve as cell level controllers, overseeing the coordination of a group of smaller PLCs. In fact, programmable controllers are fast becoming the universal building block for industrial automation [8]. More than likely, PLCs will be a major factor in the now developing computer integrated manufacturing (CIM) hierarchical architectures [7].

2. Trends in the Programmable Controller Market

2.1. Growth Statistics

The lavish praise being given to programmable controllers is substantiated by the sales growth in the PLC market. Few products have been more rapidly and widely accepted in industry than programmable controllers. For the years 1977 - 1982, sales of PLC equipment grew at an annual rate of 44% from $60 million

in 1977 to $370 million in 1982 [1]. PLC sales are currently valued at more than $600 million. Sales show no signs of slowing. By the mid '90s, PLC shipments will approach the $4 billion mark, predicts Business Trend Analysts, a market research firm based in Long Island, N.Y. [12].

Although PLCs were originally targeted for the automotive market, their ruggedness, flexibility, and ease of use have expanded their application into nearly every industry, from chemical processing to metalworking, with energy management representing the biggest market for PLCs. However, material handling equipment is moving up quickly and is expected to take over the number one spot in the next couple of years as PLCs move into the factory of the future. Over the next decade the material handling equipment market for PLCs will grow from $200 million to $650 million, predicts Business Trend Analysts. And only a small step behind will be the food processing equipment market, where PLC sales are expected to approach $640 million by 1995 [12].

2.2. Reasons for Growth

The tremendous growth in the PLC equipment market has been fueled by a number of factors. The proven reliability record of PLCs has been a contributing

factor, as well as the substantial corporate backing of established manufacturers of programmable controllers [1].

More importantly, new markets for PLCs, such as food processing and materials handling, have developed as the devices have become smaller and more powerful [1]. Alex C. Mair, vice president, General Motors Corp. reiterated this point in his address at the Programmable Controllers Conference. He commented, "New developments in electronics and computers are key to many elements of the second industrial revolution. The incredible ability of the modern computer to store information, to calculate, and to release data at tremendous speeds is the key to the exponential rise in technology we see all around us" [2]. The key to the increasing performance of the modern computer has been the development of microprocessor technology.

Increasing performance alone would not have triggered the growth in the PLC market. Performance gains were coupled with the decreasing cost of microprocessors. The affordability of PLCs justified their growing use in various industrial applications. Thus, the lower-than-ever cost/performance ratio of PLCs is the most important reason for skyrocketing sales.

2.3. Dominance of Market

An interesting note about the PLC market is that although more than a dozen manufacturers produced PLCs as of May 1984, two manufacturers, Allen-Bradley and Gould Modicon, dominated the market with a combined share of 70% [1]. Even though a number of new manufacturers have since entered the PLC market and chipped away at this market share, these two companies remain dominant in the current U. S. market.

3. Programming the Programmable Controller

3.1. Relay Ladder Logic Programming

Up to this point, it has been determined that the programmable controller is a reliable piece of hardware. But the hardware is useless without a simple programming language for writing the control program. The industry-adopted standard programming language is known as relay ladder logic programming. This language is derived from series and parallel-connected ladder networks that were drawn to wire electromechanical relay panels. Ladder networks were so named because when drawn, they look like a ladder. The left vertical post is the "hot" bus connected to a power source and the right post is the "neutral" bus. Interconnected relay contacts and coils are tied between these two posts. These circuits, drawn horizontally between the posts, are known as rungs.

When the proper set of conditions exists, power flows from the left post through the contacts to energize the coils. These coils drive the actuators that cause specified actions to occur on the controlled device. (Refer to Figure 3.1 for an example ladder network diagram.)

Although PLCs were designed to replace these hardwired relay systems, they maintained the idea of ladder networks for programming. In effect, programming the PLC was done by copying the ladder network drawings into the PLC through a graphical interface. Imaginary power flows through the software networks to energize the programmed coils. Based on the coils that are set, the PLC energizes the actuators through its output interface. This pictorial-based system makes programming of the PLC a relatively simple task for maintenance personnel accustomed to dealing with electromechanical relay systems.

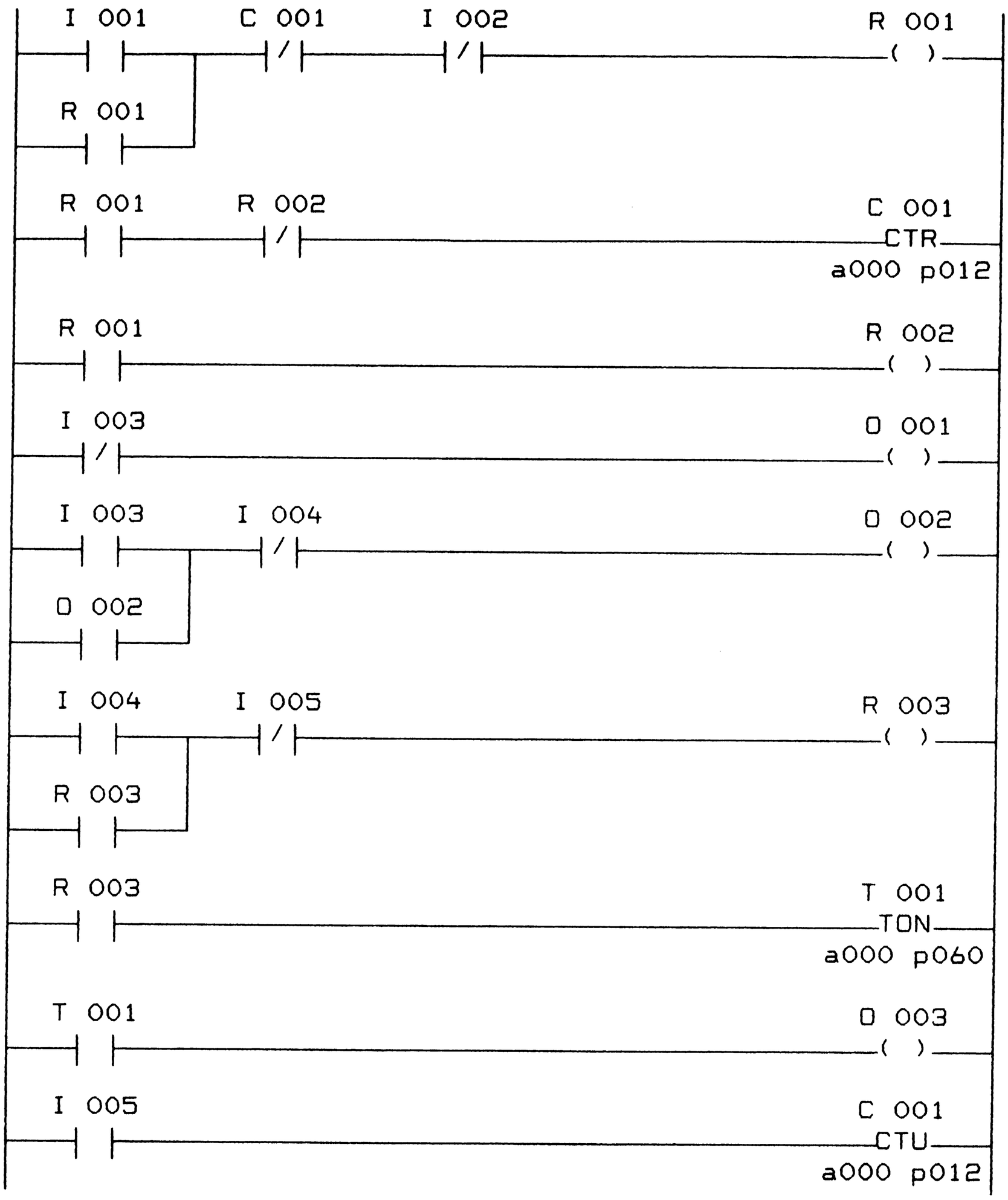


Figure 3.1. Example ladder network diagram.

3.2. Reasons to Adopt Ladder Logic Programming

It is easily implied from the previous section that the main reason for programming PLCs in ladder logic was to allow plant electricians to quickly adapt to the electronic technology, especially in applications that required replacing an existing relay design [5]. Because the electricians were very familiar with ladder networks, they could begin to use the new PLCs with a minimal amount of training.

Today, ladder logic programming is still the accepted standard. Unlike programming languages used in conventional computers, relay ladder logic is understood by most plant personnel. Says Dr. Jesse T. Quatse, vice president of technology and founder, Maxitron Corp., "Ladder language is an elegant, high-level language with beautiful graphics and a simplified, user-friendly way of expressing a logic program" [9].

Relay ladder diagrams are expected to remain popular for many more years. However, increased use of PLCs in process, integrated manufacturing, and total plant control applications is resulting in greater use of alternative languages [11]. Many of these alternative languages are based on structured programming languages such as FORTRAN and Pascal. In response to this trend, the PLC manufacturers now offer these alternative languages as enhancements to the

standard ladder logic programming language. Notice that these alternative languages are enhancements; they add to the standard ladder logic programming language. Relay ladder logic still remains the foundation for the programming of PLCs and it seems that this standard will continue. According to Mr. Bob Lyons, manager of PLC operations, Telemechanique, the electrical maintenance man will always want relay ladder languages because he's the person troubleshooting and maintaining the PLC, regardless of who else is involved in designing a PLC solution. It doesn't take a great deal of expertise and experience to maintain and troubleshoot relays [9].

In summary, relay ladder logic programming is a simple, pictorial-based language that is understood by many plant electricians and maintenance personnel and is easily learned by novice programmers.

3.3. Programming Terminals

The relay ladder logic program is entered into the PLC using a device known as a programming terminal. The first programming terminals were monochrome CRTs with industrialized casings and keyboards. These specialized terminals could be used only for the entry of ladder logic programs. In addition, programming could be accomplished only when the terminal was connected to the

PLC. Thus, programming of PLCs usually meant a trip to the plant floor.

Later, tape loaders were developed for the storage and retrieval of ladder logic programs on magnetic cassette tapes. With this development, programs could be developed at alternate locations and then carried to the plant floor on tape.

Today, there are a variety of programming terminals. Small, handheld terminals that use LCD technology can display one rung of logic at a time. These mini-terminals can be easily carried around in a briefcase and used for quick programming fixes in the plant. Large, color computers are also being used as programming terminals. The use of color makes the pictorial programming interface even simpler. The advance from terminals to actual computers had led to the development of off-line programming systems; no longer does the terminal have to be connected to the PLC to enter a program. The program can be developed off-line in the computer's memory and disk drive system, and the completed program can be either downloaded directly or transported via newer magnetic cartridge tape to the PLC.

3.4. Personal Computers as PLC Programming Terminals

The latest advance in the area of programming terminals is the use of the emerging industry standard IBM PC-compatible computers for the entry of ladder logic programs into the PLC. This trend has all but caused proprietary programming terminals and panels to become obsolete [9]. Users can now program different manufacturers PLCs on a common programming terminal. The PC systems also support off-line PLC programming which allows engineers to program in their offices and not in the midst of the plant floor distractions. In addition, dedicated programming terminals no longer sit idle because the PC can be used for a tremendous range of other functions, such as word processing, spreadsheet analysis, and project planning, when not being used as a programming station [8].

One critical function that can be performed on a PC system is the documentation of ladder logic programs. Documentation packages allow programmers to add text descriptions to each contact and coil used in the program, and commentary (similar to the comment statements in FORTRAN or Pascal programs) to describe the function of each rung of logic. These documentation systems are capable of printing I/O cross-referencing reports which are indexes that lists every rung in which a given contact or coil appears in the program. Before

these documentation systems existed, documentation was done manually on the ladder network diagrams. Manual documentation was prone to errors; therefore, it was not useful to the PLC programmer. Today's systems, which contain comprehensive labeling, comments, and I/O cross-referencing, greatly simplify the tasks of debugging and troubleshooting [6]. (Refer to Figure 3.2 for a documented ladder logic diagram and contrast the readability of this diagram to the ladder network diagram in Figure 3.1.)

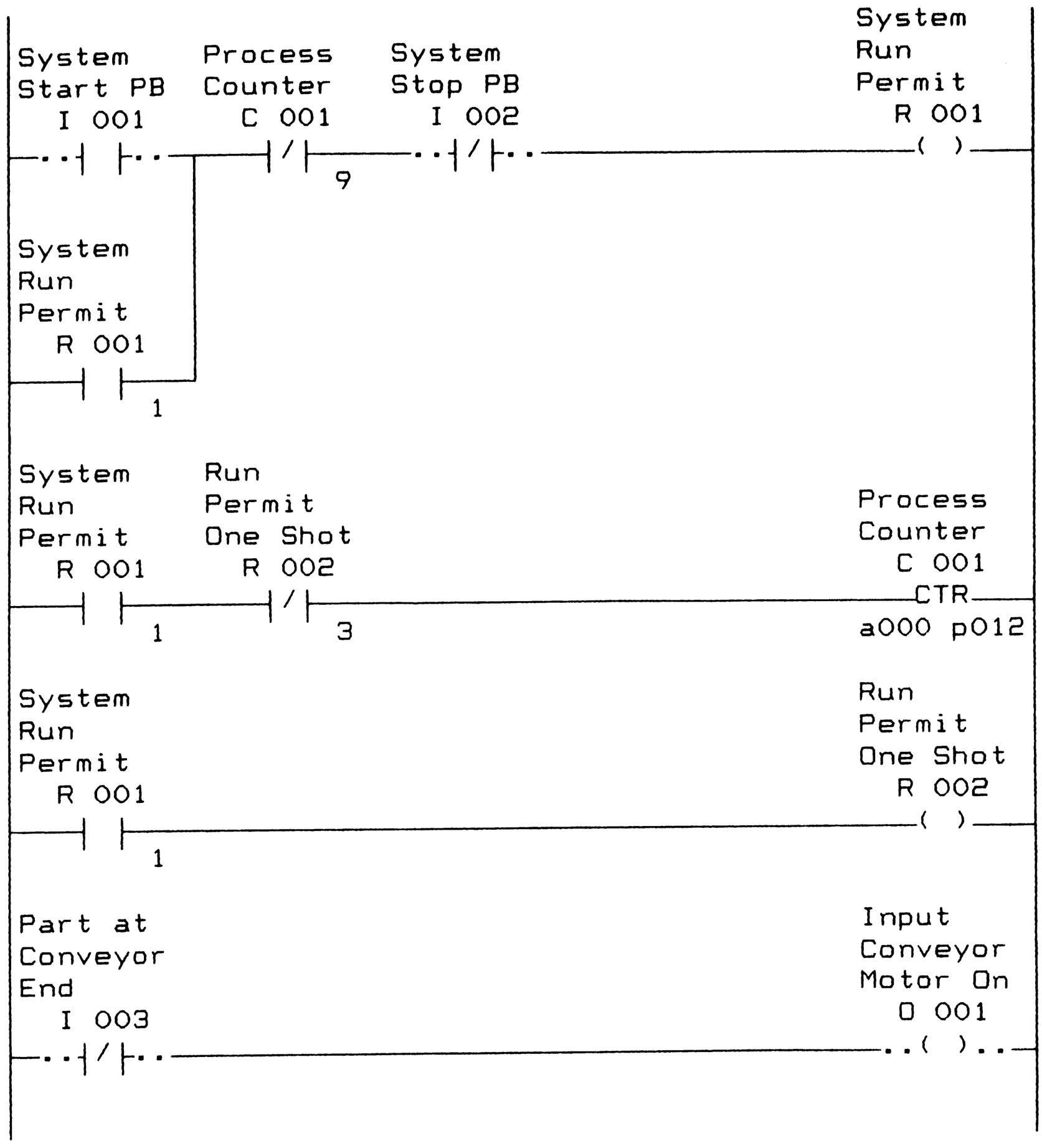


Figure 3.2. Documented ladder logic diagram.

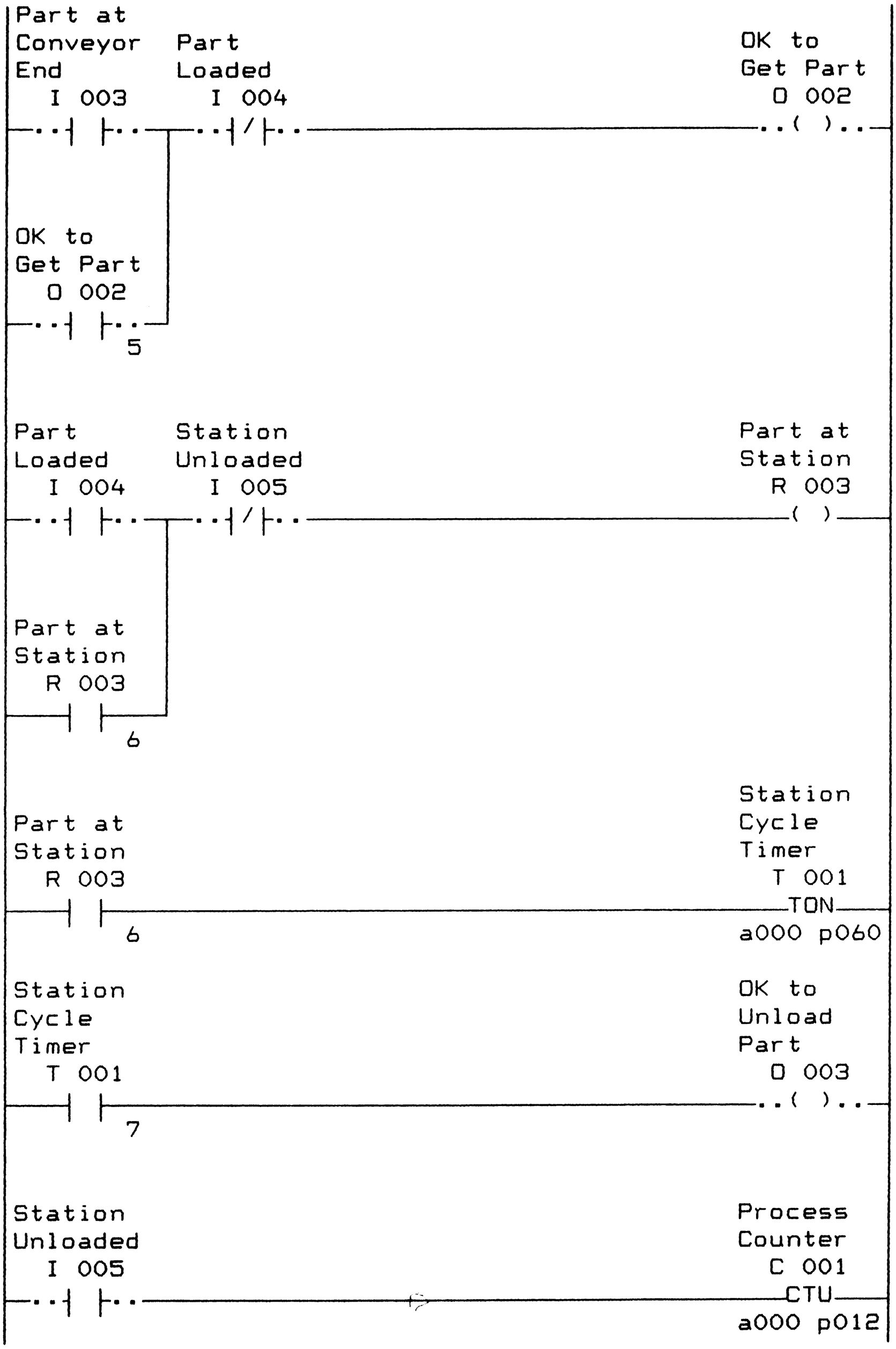


Figure 3.2 (continued). Documented ladder logic diagram.

4. Needs in Today's Programmable Controller Market

More PLCs are being used today than ever before. The tremendous growth rate has stimulated demand in two areas associated with PLC programming. These areas are the need for training tools for the many new PLC programmers and for more sophisticated programming development systems to further simplify the PLC programming task.

4.1. Training Tools for New Programmers

Many new PLC programmers are required to program the ever growing number of programmable controllers. These new programmers are being recruited from college graduate engineers. These engineers, however, do not receive intensive training in electromechanical relay control design because it is fast becoming obsolete technology. Nor do they receive ladder logic programming skills because universities focus on higher level programming languages such as FORTRAN and Pascal.

On the other hand, the manufacturing industry is demanding new-hire engineers with extensive PLC training to replace the retiring plant electricians. No longer is on-the-job training sufficient to train these new programmers. In fact, the increased functionality of today's PLCs which includes arithmetic, data manipulation, and block transfer instructions, requires

new PLC programmers to have some advanced computer programming training to become fluent in PLC programming.

For these reasons, inexpensive PLC programming training tools need to be developed, particularly for use at the college level. Preferably, these tools should be designed for the IBM PC-compatible computer which are widely accessible on today's campuses.

4.2. More Sophisticated Programming Tools

Since the experience base of PLC programmers is shrinking, one method to increase their productivity is to develop more sophisticated programming tools. The documentation and off-line programming systems that exist today are simple, yet effective programming tools. An important advantage is that some of these systems are now being developed for IBM PC-compatible computers. Documentation systems improve the readability of programs by adding text descriptions to the ladder diagrams. Off-line programming sytems allow the creation of ladder logic programs without using the PLC controller. These systems allow for the maintenance of PLC programs while the controllers remain on-line on the factory floor. Another advantage of off-line programming systems is the ability to create a database of PLC programs and subroutines on the PC's disk drive

system. This type of library allows for faster programming by reusing old programs and for standardization of PLC programs by accessing standard subroutines.

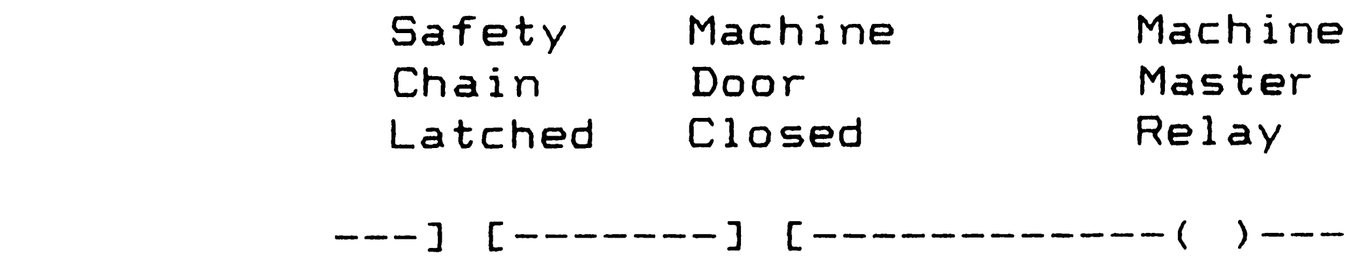
There are two drawbacks to the current documentation/off-line programming systems. First, documentation and off-line programming are sold as two separate packages that can operate on a common database of programs, but not as a totally integrated system. Separate packages allow the end user to purchase only the functionality that he requires. However, this marketing methodology forfeits advanced functions that could be implemented on an integrated system.

For example, a PLC programmer might need to implement the following logic:

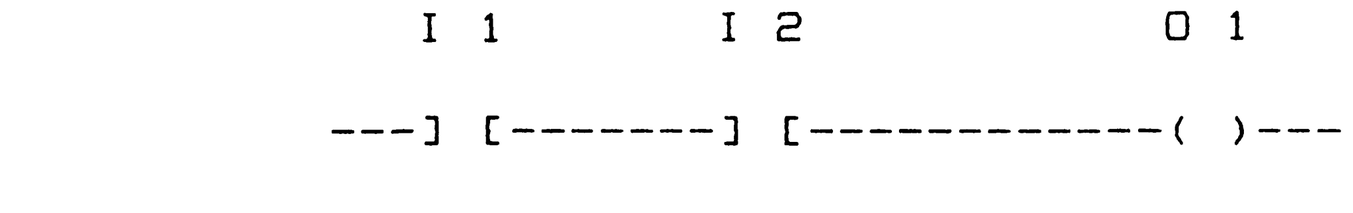
> If the safety chain is latched and the machine door is closed, then the machine can be started.

There will be two sensors to indicate that the interlocks are satisfied and the machine can only be started when a master relay is energized.

Normally, the programmer would sketch this logic onto paper and it would look as follows:



Since the off-line programmer recognizes input/output numbers and not text descriptions of the I/O points, the next step would be to assign I/O numbers to each I/O point as follows:



This rung could now be entered into the program. The final step would be to use the documentation package to reassign the text descriptions to each I/O point.

This extra work could be avoided with an integrated software package. The programmer would be able to write the entire program using text descriptions only. When the programmer is satisfied with the complete program, he could then assign all I/O numbers. Assigning I/O numbers after completing the program promotes an efficient and orderly numbering system because all required I/O points are better known when the program is completed. There is no havoc created by added, deleted,

or changed I/O points which occur frequently as the program is developed.

Another advantage of an integrated package is the text description-based search. Searching allows the programmer to scan the program for a specified I/O point. This is similar to a word processor's capability to search for words in a text document. Today's programming systems implement searches based on I/O numbers. This means that the programmer must maintain a list of the text descriptions currently assigned to each I/O number, either on paper or in his head. However, it is more natural for the programmer to ask for an I/O point by name. From the example above, the programmer might want to locate this rung in his program. He would accomplish this by searching for the rung that turns on the machine master relay. Today's off-line programmers would require the programmer to recall that this relay is assigned to 02 and then to perform a search for it. An integrated package would directly search for the output called "Machine Master Relay."

The second drawback is that today's packages do not include a program simulation capability. Instead, the makeshift solution offered is the ability to upload/download programs into the actual programmable controller for testing there. This creates two problems. First, most programmable controllers do not

have the text description capability because they were designed for high-speed industrial control and no memory was allotted for storage of "unnecessary" text. Second, and more critically, the cost to maintain a dedicated programmable controller for simulation purposes is prohibitive.

5. The Proposed Solution:

A PLC Program Development System

The original PLCs were designed as relay replacers and PLC programming was based on the existing techniques of ladder diagramming. These simple systems were designed to improve the productivity of manufacturing processes by augmenting manual labor with automation and to reduce direct labor costs. Programmable controllers have been quite successful in achieving these desired goals.

The push in today's manufacturing processes for full computer integrated manufacturing requires much more from PLCs and all programmable automation. Since the control and integration of these processes is much more demanding than simple relay replacement, more sophisticated programming tools and techniques are needed. In addition, the increasing complexity of CIM systems requires an enormous initial investment, not only in capital but also in engineering. Therefore, the

goal of the latest automation equipment has become the decrease of engineering input [3].

The solution to achieve these goals in the PLC market is to develop a PLC program development system. A PLC program development system is an integrated software package that allows for:

- o off-line programming for the development of new programs as well as the modification of existing ones,
- o documentation of these programs (I/O descriptions, section titles, and rung comments) interactively as they are developed including the ability to use symbolic text descriptions for I/O points (as discussed previously), and
- o off-line simulation capabilities (that doesn't require the actual PLC or any hardware other than the computer system that the development system runs on) for testing PLC programs.

An integrated development system of this type would allow the PLC programmer to write, debug, and modify PLC programs quickly and easily.

As mentioned previously, the optimal solution is to create a development system that is based on the widely available IBM PC-compatible computer. This accessibility makes for a cost-effective solution

because a new computer system doesn't need to be purchased. Even if PCs aren't currently available at the PLC programming site, the solution is still cost-effective because of the relative inexpensiveness of IBM PC-compatible computers.

In addition, this development system should be easy to use, commonly called user-friendly. This can be accomplished by using pull-down menus (which presents the user with a list of the options currently available to him) and context-sensitive help screens (which means that information is displayed based on the current part of the development system that is being used).

A development system based on the previous criteria becomes an effective training tool. Because the entire PLC program can be written, debugged, and modified off-line on an IBM PC, many more engineers will be able to access and learn about PLC programming. The pull-down menu and help systems will allow new users to start programming without many hours of preparation time spent leafing through cryptic computer manuals.

Finally, this PLC development system should be inexpensive. This would guarantee widespread use of the system. In particular, colleges, which are desperately in need of a PLC programming teaching tool (as discussed previously) but do not normally have funds available for

this type of expenditure, would be able to afford this development system.

6. Description of Currently Available Products

This section is not meant to be a complete listing of the available PLC program development systems. Instead, it is meant to show trends that are occurring in the marketplace, and hint at the possible direction that future programs might take.

6.1. Dedicated Computer-based Systems

One of the original PLC documentation-only systems was the Xycom Ladder Diagram Translator (LDT). LDT ran on a dedicated CP/M operating system computer. LDT allowed the PLC programmer to download the PLC program and add descriptions and comments to the ladder listing. The commented listing, as well as a number of cross reference reports, could be printed out. The programmer could then reference these separate printouts as he made modifications to the PLC program. A later modification to LDT allowed the programmer to make minor program modifications off-line and then to upload the altered program to the PLC.

There are a number of drawbacks to this system. Since the dedicated computer system was based on an 8-bit processor, LDT was very slow. In addition, LDT

could perform only one task at a time. Thus, printing of a sizable ladder listing could tie up the system for 3-4 hours. Finally, LDT was independent of the PLC. Any modification to the PLC program required the download of the new program followed by the regeneration and printing of the new documented ladder listing. Since this was a time-consuming task, regeneration would be saved until a number of modifications had been made. During the interim, the programmer had to work from outdated documentation. Often, this waiting period resulted in the failure to include all of the changes in a new program generation. The net effect of this system was the proliferation of inaccurate documentation data that resulted in a number of wasted hours for PLC programmers.

In spite of its drawbacks, LDT was the first step in the right direction for PLC programming documentation systems. In fact, it was regarded as the industry standard for a number of years. Because of this acceptance, LDT systems are still in wide use today to maintain the many existing programs that were documented using this system. However, as more of the new systems offer translators to convert LDT files, the Xycom LDT system will become obsolete.

6.2. Minicomputer-based Systems

Minicomputer-based systems such as Process & Instrumentation Design, Inc.'s ProDoc documentation system attempted to correct the slow performance of the LDT system. By using a minicomputer, tasks could be performed in parallel; documentation of a program could be performed while another program listing was being printed. In fact, tasks unrelated to PLC programming could be running on the minicomputer along with ProDoc.

The prohibitive factor for this type of system is the initial capital outlay for a minicomputer system. As the size and environment requirements for minicomputers systems lessen, greater use of ProDoc-like systems is anticipated. Another contributing factor to greater usage is the ability to attach the newer PLCs to communications networks, allowing for a direct upload/download link to the minicomputer.

6.3. Programmable Controller-based Systems

The dedicated and minicomputer-based documentation systems discussed previously were stand-alone systems developed by third party vendors. The PLC manufacturers soon realized that they needed to become part of this market. The advantage that PLC manufacturers could offer was an integrated programming and documentation system. No longer did the programmer have to look at

the PLC programming terminal while referencing a separate ladder diagram printout to decipher the program's operation.

Examples of this type of system are the Honeywell IPC 620 PLC and the General Electric Workmaster Programmable Control Information Center. These systems allowed for 7 character labels to be assigned to I/O points in addition to the actual I/O numbers. Both the number and label are displayed on the programming terminal, and either could be used to reference the I/O point. The integration of programming and documentation greatly simplified the programmer's job. As he changed the program, he could also update the documentation. Thus, documentation of PLC programs became much more reliable. In addition, the display of the documentation information along with the program made debugging much easier.

There are two drawbacks to this system. First, the amount of documentation that could be attached to the program was limited compared to the LDT and ProDoc systems. Programmable controllers are industrial workhorses when running control programs, but they are not very good at maintaining text. A more costly drawback is that these documentation systems worked only on the manufacturer's PLCs. The investment in one manufacturer's documentation system made it more costly

to switch to a different PLC better suited for new applications. This lack of flexibility is most likely the reason that these type of systems have not gained wide acceptance in today's market.

6.4. IBM PC-based Systems

Rapidly gaining acceptance today are the IBM PC-compatible computer-based systems such as the Universal Programmer/Documenter (UP/DOC) by Xcel Controls, Inc. and the Ladder Logic Documentation and Off-Line Programming System by Taylor Industrial Software. These systems offer capabilities similar to LDT or ProDoc with the advantages associated with being IBM PC-based.

Unfortunately, these systems do not yet integrate the programming and documentation systems. Thus, the advantages of having commented program displays on screen as found in the PLC-based systems are not yet available on these systems. In addition, these systems do not yet offer a simulation capability.

A recent development in the PLC market was the PLC-2 I/O Simulator by HEI Corporation. This system allows the IBM PC to be connected to an Allen-Bradley PLC 2/20 or 2/30 for simulation purposes. The user can write simulation programs in higher-level languages such as FORTRAN or Pascal on the PC and then run them to test the functionality of the PLC program. The PC in effect

becomes the machine or process to be controlled. Thus, programs can be more thoroughly debugged before the actual machine or process is connected to the PLC.

In summary, the current PLC market offers excellent off-line programming, documentation, and simulation systems based on the IBM PC. The next step is to offer these functions as an integrated PLC program development system.

7. Thesis Description

The purpose of my thesis project is to verify the feasibility of a PLC program development system. The method for verification is to actually program a working development system that meets the proposed criteria (outlined in Section 5, "The Proposed Solution: A PLC Program Development System"). The results of this project will be the specification of the data structures and program algorithms needed by an integrated PLC program development system and the demonstration that the specification can be implemented on an IBM-PC compatible computer.

The IBM PC-compatible computer should be equipped with a monochrome screen, DOS 2.0 or higher, and 256K bytes of RAM memory. The use of a hard disk drive is recommended for optimal performance. The software will

be written in Borland International's Turbo Pascal which is a powerful extension of the standard Pascal language and is now recognized as the industry standard Pascal language for the IBM PC. The PLC programming language implemented in this system will be a limited instruction set containing the more frequently used instructions, contacts, coils, timers, and counters. The method of entering the program will be based on the Allen-Bradley PLC methodology since they are the current U. S. market leader.

8. Justification for this Thesis

The justification for a PLC program development system has been described previously; the creation of an integrated PLC programming system does seem warranted. The demonstration that such a system is feasible may spurn PLC manufacturers to consider the development of this type of system. In addition, a small-scale working model of a PLC development system will help to develop user interest. It may just be pressure from these users that cause the PLC manufacturers to introduce advanced PLC program development products.

More specifically, this small-scale software package is meant to serve as a teaching aid for ladder logic programming, particularly at the collegiate level. The final product of this thesis should be capable of

serving this need; a future industrial-capable PLC development system would improve upon the system's training capabilities.

9. Programming Considerations

9.1. Turbo Pascal

The Pascal language was selected because the data structures implemented in Pascal, in particular records, linked lists, and binary trees, were most appropriate for developing the data structures required to store the PLC program data base.

Turbo Pascal was selected for a number of reasons. First, it is becoming widely accepted as the Pascal language for programming on the IBM PC. Next, it is a very inexpensive compiler; the list price is under $100. In spite of the low cost, Turbo Pascal is one of the fastest compilers on the market. Also, Turbo Pascal offers a number of extensions to the Pascal language that are geared for the IBM PC. The ones that were important to this application were:

- Standard functions and procedures for performing graphics drawing on the screen,
- An easy interface to call MS-DOS interrupts and functions without coding assembly language macros, and

- An overlay system that allows for swapping of program pieces on and off of disk drives that is transparent to the programmer (which means that it does not require special programming; instead, Turbo Pascal takes care of this function).

9.2. Menu-Driven Software

The program will be menu-driven wherever possible. Usually, these menus will appear on the last three lines of the screen. This area will be known as the "soft key" command area display. "Soft key" means that the command area display changes based on the current state of the program. For example, each function key can have up to four functions associated with it (unshifted, shifted, control, and alternate key combinations). Normally, the command area display will list the unshifted function key functions. By pressing the shift, control, or alternate key, the command area display will change to the respective menu of functions. This eliminates the need for a template or overlay (for example, Lotus 1-2-3 or Word Perfect require special templates) to remind the user of each key's functions.

Some functions will have menus that fill the entire screen. For example, the print function will display all the possible reports that can be printed and prompt

the user to select only those reports that should be printed.

9.3. Modular Design

The program will be modular in design so that functions may be added or changed with minimal difficulty. This will allow for future upgrades to the program; for example, the implementation of a different instruction would require the modification of only a few procedures. To support the modular design, global variables will be used sparingly. Instead, variables will be passed as local parameters into functions and procedures that require them. Also, constants will be used to specify program parameters. This ensures that program parameters can be easily modified throughout the program by redefining the constants in a single place at the top of the program.

10. An Integrated Database:

The Heart of the PLC Development System

The PLC development system revolves around its integrated database. The database links the program instructions with the text descriptions and comments that are associated with them. Through these integrated links, programmers can access contacts by either their

I/O numbers or their text descriptions; either form of reference is equivalent in the integrated database.

Three types of data structures form the integrated database. They are the program storage, data tables, and the description tree. Links between these data structures allow information from one data structure to be referenced by another. For example, the program storage data structure references the description tree to get the text description names for each of the contact and coil instructions in the ladder logic program. The Pascal definitions and further details concerning these data structures can be found in Appendix A.

The PLC ladder logic program is constantly changing. For example, the programmer adds, modifies, and deletes instructions and rungs of the program many times during the debug stage. The program storage data structure reflects this constantly changing nature. It uses linked lists to allow for dynamic allocation for program storage space on an "as needed" basis. Doubly linked lists are used to allow for movement through the database in either direction, for example, to advance or to go back a number of rungs of the program. The program storage data structure (diagrammed in Figure 10.1) consists of a main trunk of doubly linked nodes (shown on the left in Figure 10.1).

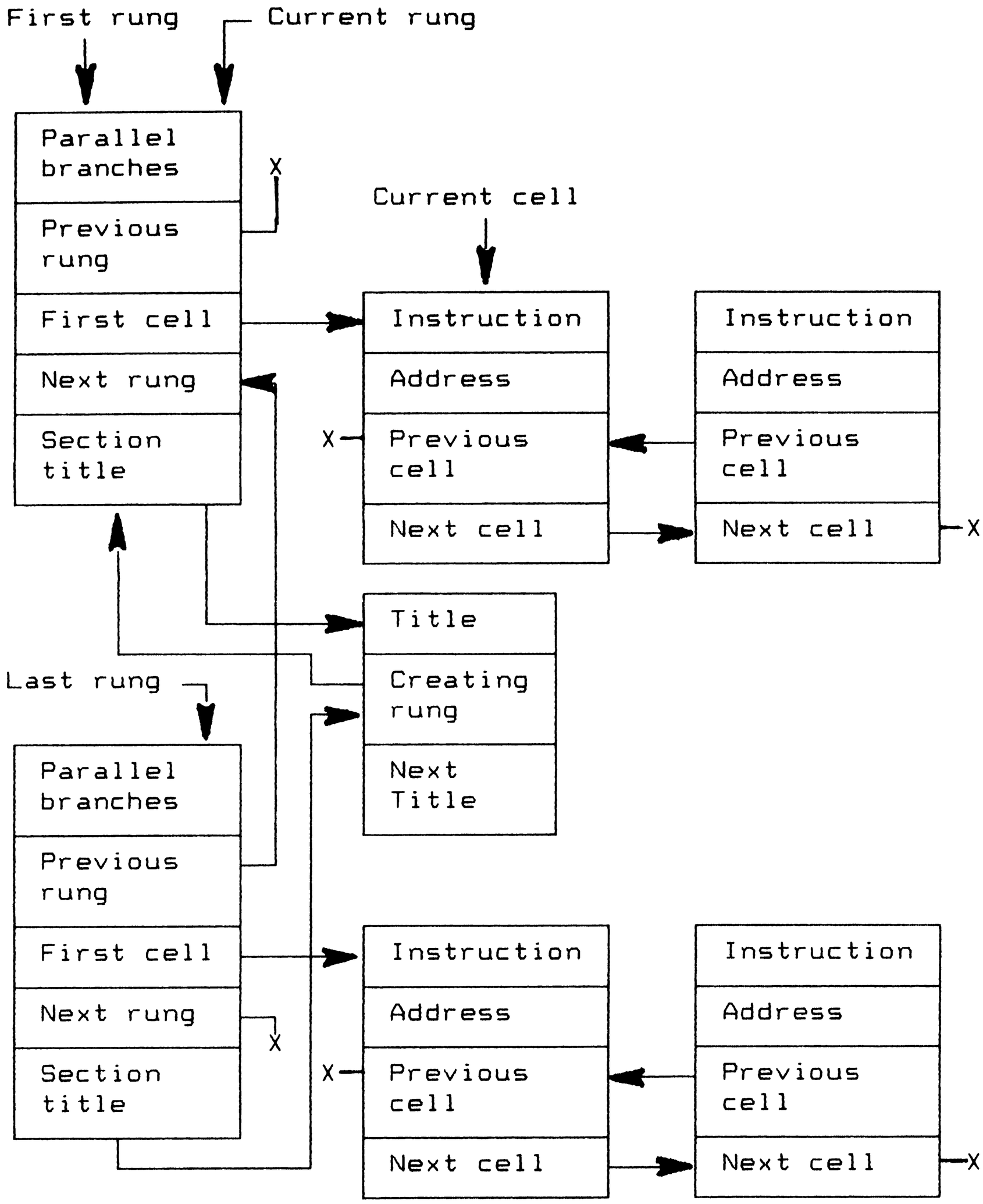


Figure 10.1. Data structures for program storage.
Boxes represent records of data.
X represents a pointer to nil.

Each node (represented by the larger boxes in Figure 10.1) represents one rung of the ladder logic program. The double linking is accomplished with "previous rung" and "next rung." Each of these rung header nodes in turn points (with "first cell") to a doubly linked list. This secondary linked list stores the instructions for that rung. Each instruction node (represented by the smaller, square boxes in Figure 10.1) stores one program instruction, such as a normally open contact or a timer. Notice the double linking implemented with "previous cell" and "next cell." A number of pointers are used to mark specific locations in the program data structure. These are "first rung" which marks the beginning of the program, "last rung" which marks the end of the program, "current rung" which points to the rung which is currently at the top of the display screen, and "current cell" which marks the instruction on which the cursor is currently displayed.

The data tables contain information about each I/O point that could be used in the PLC program. The data tables store cross-referencing information, every rung number in which a given I/O point is used. It stores the presets for all timers and counters. It also maintains run time information for use during simulation testing. This run time information is the on/off status

of contacts and coils, the time elapsed for timers, and the current count for counters.

The description table (refer to Figure 10.2) stores an alphabetical listing of the currently used text descriptions for all program instructions in a data structure known as a binary tree. The tree is arranged in alphabetical order such that all descriptions stored in the left subtree of a given node are alphabetically before the description stored in the given node and all descriptions stored in the right subtree of a given node are alphabetically after the description stored in the given node. By constructing this tree such that it remains height balanced, text descriptions can be found very quickly in this database. One example of the need for the description tree is searching for a contact by description. The desired contact description is found in the tree. Then, the description's link to the program storage data structure is used to locate the rung in which the instruction is contained. For a more in-depth discussion of height balanced binary trees, refer to the reference by D. Knuth [10].

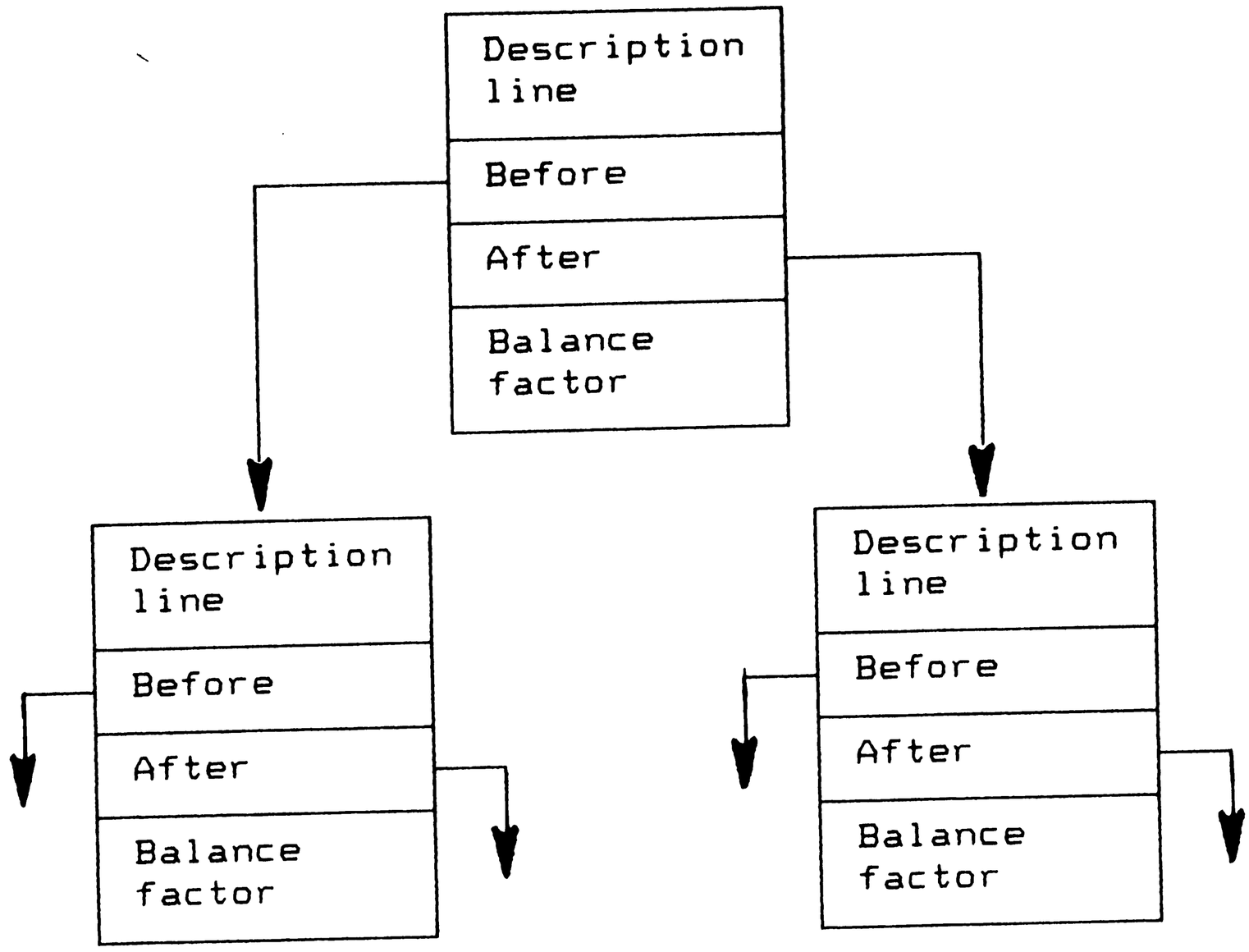


Figure 10.2. Description tree.
Boxes represent records of data.
X represents a pointer to nil.

11. Current Status of the Program

The "User's Manual" section (Section 13) provides a detailed description of the currently implemented functions. This section discusses the software development that has been completed as part of this thesis project.

Currently, the program is capable of retrieving a program file from disk and loading it into the development system's internal data structures. The ladder logic diagram, including text descriptions and rung cross reference numbers on contacts, can be

displayed on the screen. Cursor keys allow for movement throughout the program. Some of the search capabilities have been implemented. A filer function for retrieving, saving, deleting, renaming, and listing a directory of program files is functional, as well as the print function for printing a hard copy of the PLC ladder listing program to a dot matrix or daisy wheel printer. Commenting of the program with section titles and a program identification sheet is also available.

These functions listed above are implemented in fully commented Pascal program files which total over 9000 lines. Approximately 3000 of these lines are actual executable code. These files compile to a program size of 51K bytes. Development of these program files took seven months of work averaging 12 man-hours per week.

Functions that still need to be integrated are the entry, editing, and deletion of new PLC program rungs through the keyboard, the cut/paste rung function, some of the search functions, some of the supplemental print reports, and the simulation function. Algorithms to perform these functions have been developed.

The reason these functions have not yet been integrated is that the program size limit of the Turbo Pascal compiler has been exceeded. Approximately 8K bytes of program code remains to be integrated. This

memory overflow situation was not anticipated because Turbo Pascal claims to have a 64K byte program size limit. However, closer examination revealed that the Pascal run time library requires 13K bytes of this space, thereby reducing the effective program size limit to 51K bytes.

12. Future Direction for the Program

12.1. Short Range

The program memory limitation can be circumvented through the use of the overlay system. By keeping part of the program on disk, the program memory limit will not be exceeded. Storing of parts of the program on disk does reduce the performance of the program because disk accesses must be performed to get those pieces of the program when they are needed. With careful planning, the amount of disk accesses, and therefore the degradation of the program performance, can be minimized.

The use of the overlay system will require some restructuring of the current program. Because of the modular nature of the program, there are some natural breakpoints in the program for segregating the pieces to store to disk. It is anticipated that this restructuring (with program debug) will take approximately two months to complete. Integration of

the remaining program functions is anticipated to take an additional one month.

However, the results to this point indicate that a PLC development system for the IBM PC is feasible.

12.2. Long Range

As this program was developed, a number of new ideas were envisioned for future revisions of the system. These long range goals are intended to upgrade the current prototype into a workable product for industrial users.

The obvious enhancement is the extension of the instruction set to ones that are used by currently available PLCs such as the Allen-Bradley PLC-2/30 or the PLC-5. Because the program is modular in design, this extension is feasible. Limitation on any extension would be the number of functions that can be attached to the function keys, currently limited to 40 minus the number of keys that are used for functions other than program instructions. A new menu-type entry system may be needed to overcome this limit.

Another enhancement would be the ability to upload and download PLC programs to and from the programmable controller. This would require a simple program translator that converts the PLC program from the

internal storage system of the development system to the coding format used by the PLC memory.

Finally, a major enhancement to the system would be the development of a batch analysis simulation function. This function would analyze an entire PLC program and, based on a set of internal rules, would print out timing charts of how the program will operate. It is envisioned that this function would be developed using an artificial intelligence language such as LISP or Prolog. A natural selection here could be Turbo Prolog, but further investigation of this solution is required.

12.3. What Should Be Done Differently?

The PLC development system that was developed in this thesis might require a major overhaul if the current program cannot be structured to work as an overlay system. The modular nature of the program should allow for this restructuring, so drastic revisions are not anticipated. In reference to this possibility, it is appropriate to discuss what could have been done differently. Better documentation from the outset would have dramatically improved the development of the program. The use of the structured analysis method to initially plan and then maintain the interaction of all the program's subroutines would have been a great benefit. This method was not elected for

use because the complexity of the program was underestimated from the start. It was felt that the full commentary in the program source files would be sufficient. However, the increasing length of the program and the longer time frame needed for programming warranted the use of structured analysis to better document the program.

Another suggested improvement to the documentation scheme would have been the maintenance of a more complete list of program revisions. Since there was only one programmer, it was felt that a complete revision list would be an unnecessary and time-consuming task. Many of the program changes were documented in the program's comments. However, as the program progressed, it was soon apparent that a more formal list of revisions would have contained beneficial information.

13. User's Manual

Since the PLC program is not yet completed, this section will describe the use of the completed functions and specify the desired operation of the functions still to be implemented. The current status of each command will be detailed in the first paragraph of its description. In addition, this user's manual will

discuss the reasons why functions were implemented using the described scheme.

13.1. Program Start up

The commands in this section have been implemented and debugged, and are currently available for use.

The PLC development system is started by typing (at the MS-DOS "A>" prompt) the command:

A>stp.

13.1.1. "/g" Option

There are two optional switches that can be used when starting the program. The "/g" option tells the PLC development system that a color graphics card is installed in the computer. Otherwise, the program defaults to a monochrome card. This option is invoked by typing:

A>stp /g or A>stp /G.

13.1.2. "/d" Option

The "/d" option sets the PLC program for printing to an attached daisy wheel printer. This causes only standard ASCII characters to be used for printing and not the special higher-order box drawing characters available in the IBM character set. The program defaults to the dot matrix printer setting which uses

the higher-order character set. This option is invoked by typing:

A>stp /d or A>stp /D.

This printer setting can also be changed from within the program. Refer to the Print function for details.

13.1.3. Using Both Options

Both options can be selected for use by typing:

A>stp /g /d or stp /d /g.

Note that the order for invoking the options is not important. However, the space between the two option switches is necessary.

After typing the appropriate command, a boot screen will appear and some music will play. At this point, press the Space Bar to enter the PLC development system main menu screen.

13.2. Main Menu Screen

The main menu screen consists of the status display line, the program area, and the command display area.

The status line, located at the top of the screen, contains current information about the state of the program. The left part of the line displays the current rung number and the total number of rungs in the program. The center part of the line is a real-time clock. The right part of the line displays the settings

of the Caps Lock, Num Lock, and Insert/Overwrite mode. Refer to the section on Entering a Rung (Section 13.5) for a description of the Insert/Overwrite mode.

The command display area, located on the bottom three lines of the screen, displays a menu of the current functions of the keys F1-F10. By depressing and holding the Shift, Ctrl, or Alt key, this area will change to display a menu of the functions associated with the Shift F1-F10, Ctrl F1-F10, or Alt F1-F10 keys, respectively, if functions are currently associated with these keys. Refer to Figure 13.1 for the main menu function key assignments.

The remaining center of the screen is used for drawing the ladder logic program.

F1	F2
-] [- -CTU- Identification Sheet Help	-]/[- -CTD-
F3	**F4**
-( )- -CTR- Section Title Print	Comment Filer
F5	**F6**
-(L)- Simulation	-(U)-
F7	**F8**
-TON- Cut/Paste	-TOF- Address Search
F9	**F10**
Branch Start Coil Search Descriptions On/Off	Branch End Description Search Quit

Figure 13.1. Function key assignment.
Line 1 shows unshifted key function.
Line 2 shows shifted key function.
Line 3 shows Ctrl key function.
Line 4 shows Alt key function.

13.3. Quit

The commands in this section have been implemented and debugged, and are currently available for use.

To exit from any submenu back to the main menu, to abort any operation (except as noted below), or to exit the program from the main menu, press the Escape (Esc) key. One exception to this rule occurs in the Filer function. To abort a file retrieve, save, delete, or rename, type a Return only at the Filename: prompt.

In addition, a number of menus have a quit function attached to a function key, normally F10, or Alt-F10 at the main menu. This method of exit is equivalent to pressing the ESC key.

13.4. Ladder Logic Programming Instruction Set

The ladder logic program (see Appendix A for an example) can be built from the following instruction set:

-] [- Normally open contact

Power can flow when the associated coil is energized (ON).

-]/[- Normally closed contact

Power can flow when the associated coil is deenergized (OFF).

-( )- Output or internal control relay coil

Energized (ON) if a continuous path of power flow exists from the left ladder to the coil.

-(L)- Latch output

Output that remains energized (ON) even if the continuous path of power flow from the left ladder no longer exists.

-(U)- Unlatch output

A continuous path of power flow from the left ladder to this coil deenergizes a latched output.

-TON- Timer On

Continuous power flow from the left ladder to this coil for the preset period of time causes a timeout contact to energize. The timeout bit is deenergized at all other times. If the continuous power flow is broken, the timer resets to zero.

-TOF- Timer Off

Continuous power flow from the left ladder to this coil for the preset period of time causes a timeout contact to deenergize. The timeout bit is energized at all other times. If the continuous power flow is broken, the timer resets to zero.

-CTU- Up Counter

Counter is incremented by 1 when power flow first energizes this coil. Maintaining power flow to this coil will not cause additional increments. The next increment will occur only when power flow to this coil is broken and then energized again. The counter will not increment higher than its preset value. The countout bit is energized when the counter value reaches its preset value.

-CTD- Down Counter

Counter is decremented by 1 when power flow first energizes this coil. Maintaining power flow to this coil will not cause additional decrements. The next decrement will occur only when power flow to this coil is broken and then energized again. The counter will not decrement if the counter value is zero.

-CTR- Reset Counter

Energizing this coil resets the counter value to zero.

Br.	Branch Start
Start	Initiates the start of a parallel branch. A maximum of three parallel branches can be programmed per rung.
Br.	Branch End
End	Connects a parallel branch to the previous branch.

Refer to Appendix C for a description of the internal coding of the instruction set.

Addressing consists of two parts, the address type and the address number. The address type can be:

I for real inputs,

R for internal relays,

O for real outputs,

T for timers, and

C for counters.

The address number can range from 1 to 256 for real inputs, internal relays, and real outputs, and from 1 to 32 for timers and counters. Refer to Appendix C for a description of the internal coding of the address.

13.5. Entering a rung

This function is currently not implemented. The method of programming rungs was based on Allen-Bradley PLC-2 family ladder logic programming because of the current popularity of these PLCs.

Entering a rung consists of the entry of each instruction of the rung in a specific order. Each instruction of the instruction set can be accessed using the F1 - F10 and the Shift-F1 - Shift-F3 keys from the main menu.

There are two types of cursors in the PLC development system. The first will be called the cursor or the current cursor. This cursor is a large, reverse video block that indicates what the current instruction is. This block will highlight the current instruction's text description, address, graphical symbol, and cross-reference number. The second cursor is called the edit cursor. This cursor is a small blinking underscore that is used during entry and editing of text descriptions, addresses, and presets of single instructions. This cursor underscores one character at a time.

13.5.1. Entering a Contact Instruction

To enter a contact instruction, press the appropriate function key, F1 for normally open or F2 for normally closed. If the insert/overwrite mode is set to insert (by pressing the "+" key until "Insert" appears on the status line), then the graphical symbol will be drawn at the position to the right of the current cursor. If the mode is set to overwrite (by pressing the "+" key until "Overwrite" is displayed on the status

line), then the instruction at the current cursor will be overwritten by the new instruction.

The next step is to enter the text description. There are three lines with eight characters each available for the description. Notice that the edit cursor will wrap to the beginning of the next line if the end of the previous line is reached. Press Return when finished entering the text description.

If the text description has been used previously, then its corresponding address will be entered automatically. The programmer can change this address. Note that this change will reassign this new address to all instructions in the program with the corresponding text description. If the text description is new, then the address will be filled with the first available internal relay address number. First available is defined as the lowest address that has not yet been used anywhere else in the program. The edit cursor will be positioned on the address type letter. The default address can be accepted by pressing Return at this point.

Otherwise, the programmer can alter the address in two ways. First, the programmer may change the address type letter from "R" to the appropriate letter, "I", "O", "T", or "C". In this case, the address number will automatically change to the first available address

corresponding to the new address type letter. The programmer can now opt to accept this default address number by pressing Return. If no change to the address type letter is desired, then the programmer can press the Space Bar to advance the edit cursor to the address number.

The programmer can change the address number to another value and then press Return to enter it.

Note that the left and right arrow keys can also be used to switch back and forth between the address type letter and the address number.

13.5.2. Entering an Output Coil

To enter a standard output coil instruction, press F3 for a coil, F5 for a latch coil, or F6 for an unlatch coil. The graphical symbol will be placed at the rightmost part of the screen connecting to the right ladder and overwrite any existing coil instruction, regardless of the insert/overwrite mode.

The next step is to enter the text description. There are three lines with eight characters each available for the description. Notice that the edit cursor will wrap to the beginning of the next line if the end of the previous line is reached. Press Return when finished entering the text description.

If the text description has been used previously, then its corresponding address will be entered automatically. The programmer can change this address. Note that this change will reassign this new address to all instructions in the program with the corresponding text description. If the text description is new, then the address will be filled with the first available internal relay address number. First available is defined as the lowest address that has not yet been used anywhere else in the program. The edit cursor will be positioned on the address type letter. The default address can be accepted by pressing Return at this point.

Otherwise, the programmer can alter the address in two ways. First, the programmer may change the address type letter from "R" to the letter "O". In this case, the address number will automatically change to the first available output address. The programmer can now opt to accept this default address number by pressing Return. If no change to the address type letter is desired, then the programmer can press the Space Bar to advance the edit cursor to the address number.

The programmer can change the address number to another value and then press Return to enter it.

Note that the left and right arrow keys can also be

used to switch back and forth between the address type letter and the address number.

13.5.3. Entering a Timer Coil

To enter a timer coil instruction, press F7 for a timer on coil, or F8 for a timer off coil. The graphical symbol will be placed at the rightmost part of the screen connecting to the right ladder and overwrite any existing coil instruction, regardless of the insert/overwrite mode.

The next step is to enter the text description. There are three lines with eight characters each available for the description. Notice that the edit cursor will wrap to the beginning of the next line if the end of the previous line is reached. Press Return when finished entering the text description.

If the text description has been used previously, then its corresponding address will be entered automatically. The programmer can change this address. Note that this change will reassign this new address to all instructions in the program with the corresponding text description. If the text description is new, then the address will be filled with the first available timer address number. First available is defined as the lowest address that has not yet been used anywhere else in the program. The edit cursor will be positioned on

the address number. The default address can be accepted by pressing Return at this point.

Otherwise, the programmer can alter only the address number for timer instructions. The programmer can accept this default address number by pressing Return. The programmer can change the address number to another value and then press Return to enter it. The edit cursor will now be positioned for entry of the timer preset value.

If the timer address is currently used elsewhere in the program, then the preset will be filled in with the currently assigned preset value. If the programmer elects to change this preset, then the value will be changed throughout the program. If the timer address is new, then no default value will be displayed. Timer preset values are accepted by pressing Return.

13.5.4. Entering a Counter Coil

To enter a Counter coil instruction, press Shift-F1 for a counter up coil, Shift-F2 for a counter down coil, or Shift-F3 for a counter reset coil. The graphical symbol will be placed at the rightmost part of the screen connecting to the right ladder and overwrite any existing coil instruction, regardless of the insert/overwrite mode.

The next step is to enter the text description. There are three lines with eight characters each available for the description. Notice that the edit cursor will wrap to the beginning of the next line if the end of the previous line is reached. Press Return when finished entering the text description.

If the text description has been used previously, then its corresponding address will be entered automatically. The programmer can change this address. Note that this change will reassign this new address to all instructions in the program with the corresponding text description. If the text description is new, then the address will be filled with the first available counter address number. First available is defined as the lowest address that has not yet been used anywhere else in the program. The edit cursor will be positioned on the address number. The default address can be accepted by pressing Return at this point.

Otherwise, the programmer can alter only the address number for counter instructions. The programmer can accept this default address number by pressing Return. The programmer can change the address number to another value and then press Return to enter it. The edit cursor will now be positioned for entry of the counter preset value.

If the counter address is currently used elsewhere in the program, then the preset will be filled in with the currently assigned preset value. If the programmer elects to change this preset, then the value will be changed throughout the program. If the counter address is new, then no default value will be displayed. Counter preset values are accepted by pressing Return.

13.5.5. Entering a Branch Instruction

The branch start instruction can be entered by pressing the F9 key to signify that the beginning of a parallel branch will be at the current cursor. The branch start instruction will always be inserted to the right of the current cursor, regardless of the insert/overwrite mode.

The branch end instruction can be entered by pressing the F10 key to rejoin a parallel branch to the previous one. The branch end instruction will always be inserted to the right of the current cursor, regardless of the insert/overwrite mode.

13.5.6. Example Rung Entry

Refer to the first rung of Figure 3.2 throughout the following example on how to enter a rung of ladder logic into the PLC development system.

This example rung would be entered instruction by instruction using the previous descriptions of instruction entry. The order for entry would be as follows:

- Branch start (F9) to initiate the first parallel branch.
- Normally open contact (F1). Enter the text description. Note that the default address will appear as R 001. Change the R to I and then press Return to accept this address.
- Branch start (F9) to move to the second parallel branch.
- Normally open contact (F1). Enter the text description. Accept the default address R 001.
- Branch end (F10) to rejoin the two branches.
- Normally closed contact (F2). Enter the text description. Note that the default address is R 002. Change the R to C and the address number will change to 001. Accept this default value.
- Normally closed contact (F2). Enter the text description. Note that the default address is R 002. Change the R to I. The address number will remain 002. Accept this default value.

- Output coil (F3). Note that the connection line is drawn automatically and the output coil appears next to the right ladder. Enter the text description. The default address will appear as R 001 because this description exists. Accept this default value.

13.6. Cursor keys

The commands in this section except for the "Go to rung number" function have been implemented and debugged, and are currently available for use.

Refer to Figure 13.2 for the cursor key assignment.

The cursor keys allow for movement among instructions on the current rung, or among rungs in the program.

Movement among instructions on the current rung can be accomplished by pressing the following keys:

13.6.1. Right Arrow

The cursor is moved one instruction to the right. Notice if the cursor is at the end of a parallel branch, then it will move down to the beginning of the next branch. If the cursor is at the last instruction of the rung, depressing this key will not cause an error but no movement will be performed.

<--	Num Lock		Scroll Lock	
delete cell	toggle cursor/ number keypad			

	Home	Up Arrow	PgUp	-
Enter	first cell	up 1 rung	up 5 rungs	
edit cell	first rung			
	Left Arrow	5	Right Arrow	
	left 1 cell		right 1 cell	
	left 3 cells		right 3 cells	
*	End	Down Arrow	PgDn	+
go to rung no.	last cell	down 1 rung	down 5 rungs	insert/ typeover
	last rung			

Ins	Del
insert rung before/ after current rung	delete current rung from program

Figure 13.2. Cursor and Edit key assignment.
Upper box shows the key function.
Lower box shows the Ctrl key function.

13.6.2. Ctrl-Right Arrow

The cursor is moved three instructions to the right. Notice if the cursor is at the end of a parallel branch, then it will move down to the next branch. If movement of the cursor is attempted beyond the last instruction of the rung, no error will be displayed and the cursor will be placed at the last instruction in the rung.

13.6.3. Left Arrow

The cursor is moved one instruction to the left. Notice if the cursor is at the beginning of a parallel branch, then it will move up to the end of the previous branch. If the cursor is at the first instruction of the rung, depressing this key will not cause an error but no movement will be performed.

13.6.4. Ctrl-Left Arrow

The cursor is moved three instructions to the left. Notice if the cursor is at the beginning of a parallel branch, then it will move up to the next branch. If movement of the cursor is attempted beyond the first instruction of the rung, no error will be displayed and the cursor will be placed at the first instruction in the rung.

13.6.5. Home

The cursor is moved to the first instruction in the current rung.

13.6.6. End

The cursor is moved to the last instruction in the current rung.

Movement among rungs in the program can be accomplished by pressing the following keys:

13.6.7. Up Arrow

The current rung is moved up (towards the beginning of the program) by one rung and the cursor is positioned on the first instruction of the rung. Notice if the cursor is at the first rung of the program, depressing this key will not cause an error but no movement will be performed.

13.6.8. Ctrl-Up Arrow

The cursor is moved to the fifth rung before the current rung and the cursor is positioned on the first instruction of the rung. If movement of the cursor is attempted beyond the first rung of the program, no error will be displayed and the cursor will be placed at the first instruction in the first rung of the program.

13.6.9. Down Arrow

The current rung is moved down (towards the end of the program) by one rung and the cursor is positioned on the first instruction of the rung. Notice if the cursor is at the last rung of the program, depressing this key will not cause an error but no movement will be performed.

13.6.10. Ctrl-Down Arrow

The cursor is moved to the fifth rung after the current rung and the cursor is positioned on the first instruction of the rung. If movement of the cursor is attempted past the last rung of the program, no error will be displayed and the cursor will be placed at the first instruction in the last rung of the program.

13.6.11. Ctrl-Home

The cursor is positioned on the first instruction of the first rung in the program.

13.6.12. Ctrl-End

The cursor is positioned on the first instruction of the last rung of the program.

13.6.13. * (PrtSc)

The cursor is positioned on the first instruction of the specified rung in the program. When this key is pressed, the prompt "Go to Rung No." will appear in the command display area. Enter the number of the specified rung and press Return. If the specified number is less than one or greater than the total number of rungs in the program, a beep will be sounded and the prompt will be redisplayed for the entry of a new number. To abort the "Go To", press Return without entering a number and the cursor will remain at its previous position.

13.7. Edit keys

The Insert rung command has been implemented and debugged, and is currently available for use. All other functions are currently not implemented.

Refer to Figure 13.2 for the edit key assignment.

The edit keys can be used to insert instructions or rungs, to delete instructions or rungs, and to edit instructions in the program.

13.7.1. Insert an Instruction

To insert an instruction after the cursor in the current rung, set the insert/overwrite mode to insert. Confirm this by checking that the word "Insert" is displayed in the status line at the top of the screen.

Follow the procedure for entering an instruction as outlined in "Entering a Rung".

13.7.2. Insert a Rung

The Ins key allows for the insertion of a rung before or after the current rung. After pressing the Ins key, a status line will appear in the command display area. Use the left and right arrow keys, or press "B" or "A" to select insertion before or after, respectively, the current rung. The choice will be highlighted on the status line. Press "R" or the Return key to confirm the insertion. Press "E" or the ESC key to abort the operation. The inserted rung will be displayed as start and end branches on the ladders. Follow the procedure in "Entering a Rung" to add the instructions for the inserted rung.

13.7.3. Delete an Instruction

The Backspace key allows for the deletion of the current instruction highlighted by the cursor. Press the Backspace key and the confirmation prompt "Are you sure?" will appear in the command display area. Press "Y" followed by Return to confirm the deletion. Just press Return to abort the deletion.

13.7.4. Delete a Rung

The Del key allows for the deletion of the current rung. Press the Del key and the confirmation prompt "Are you sure?" will appear in the command display area. Press "Y" followed by Return to confirm the deletion. Just press Return to abort the deletion.

13.7.5. Edit a Rung

The Return key allows for the editing of the current instruction. Press the Return key to enter the Edit mode. The arrow keys will now move the edit cursor among the text description and the address. If the insert/overwrite mode is set to insert, then any modifications will be inserted at the edit cursor. If the mode is overwrite, then the modifications will overwrite the old information at the edit cursor. Press Return to exit the edit mode, or press ESC to abort the changes and restore the instruction to its state before the edit mode was entered. Note that any changes to the text description or address will be made throughout the program.

13.8. Searches

The Coil Search command has been implemented and debugged, and is currently available for use. The

Address and Description Search commands are currently not implemented.

13.8.1. Address Search

Address search moves the cursor to a new instruction in the program by specifying the desired instruction's address. This function can be accessed by pressing Shift-F8 from the main menu. At this point, the choices Before or After will appear in the command display area. Press B to look for the first occurrence of the specified address before the current rung or press A to look for the first occurrence of the specified address after the current rung. A default address, the last address that was searched for, will now appear. Just press Return if this is the desired address for which to search. Otherwise, type in the address, both address type letter and address number, of the instruction for which to search. The program display will change such that the rung at the top of the screen will contain the instruction with the specified address and the instruction will be highlighted by the cursor. If the specified address is not found, then an error message will beep and appear in the command display area. Press the Space Bar to acknowledge the error and continue. In the case of an error, the cursor will remain at its previous position.

Address search is the least useful method of searching because it requires the programmer to remember all of the address assignments in the program. Use of this function will be easier if the programmer has a copy of the cross-reference list which lists these assignments. See the Print function for details.

13.8.2. Description

Description search moves the cursor to a new instruction in the program by specifying the desired instruction's text description. This function can be accessed by pressing Shift-F10 from the main menu. At this point, the choices Before or After will appear in the command display area. Press B to look for the first occurrence of the specified description before the current rung or press A to look for the first occurrence of the specified description after the current rung. A default description, the last description that was searched for, will now appear. Just press Return if this is the desired description for which to search. Otherwise, type in the text description of the instruction for which to search. This search is normally case insensitive which means that differences in descriptions due to upper and lower case letters will be ignored. To maintain case sensitivity in the search, add "/e" at the end of the description to force a search

for an Exact match. The program display will change such that the rung at the top of the screen will contain the instruction with the specified text description and the instruction will be highlighted by the cursor. If the specified address is not found, then an error message will beep and appear in the command display area. Press the Space Bar to acknowledge the error and continue. In the case of an error, the cursor will remain at its previous position.

Searching by description is a more natural method for PLC programmers. It is easier to remember an instruction by its text description than by its address number. For example, the "Pad Pickup Counter" is easier to remember than C 010. This ease results in faster search times for the PLC programmers.

13.8.3. Coil

Coil search moves the cursor to the coil in the program that energizes the contact on which the cursor currently sits. This function can be accessed by pressing Shift-F9 from the main menu. At this point, the nearest occurrence of the coil that energizes the contact will be found. The program display will change such that the rung at the top of the screen will contain the specified coil highlighted by the cursor. If the cursor is not currently on a contact instruction, or the

specified coil is not found, then an appropriate error message will beep and appear in the command display area. Press the Space Bar to acknowledge the error and continue. In the case of an error, the cursor will remain at its previous position.

This function is useful during the simulation phase. If the PLC programmer discovers that the current rung is not being energized because of a particular contact, then he can use the coil search to quickly move to the rung that energizes the contact. Tracing through the program using this method will simplify the debug process.

13.9. Cut/Paste

This function is currently not implemented, and a specification for implementation has not yet been developed.

The Cut/Paste function is similar to the Cut/Paste function in any word processor. By pressing the Shift-F7 key, the PLC programmer can select rungs and move them from one part of the program to another. Since the rung execution order in a ladder logic program is critical, this function is quite valuable during the program debug stage if only a rung reordering is required to correct the problem. Without this function, the programmer would have to delete the rungs, and then

redefine them in the new location. Thus, this function can save the programmer a great deal of time.

13.10. Documentation

The Identification Sheet and Section Title commands have been implemented and debugged, and are currently available for use. The Comment function is currently not implemented.

The three documentation functions add text comments to the ladder logic program. This added information is useful when the program is being debugged. More importantly, these descriptions are useful when someone other than the PLC programmer, for instance, the plant maintenance electrician, needs to look at and understand the program.

13.10.1. Identification Sheet

The program identification sheet contains information about the ladder logic program. Figure 13.3 shows the list of the information in the ID sheet. The identification sheet can be viewed and modified by pressing the Ctrl-F1 key.

Use the Up and Down cursor keys to change information.
Date and revision number are updated automatically.
Press ESC when selection is completed.

Program name:

Programmer :

Project   :

Company   :

Last update :

Revision no.:

Figure 13.3. Program identification sheet menu.

The up and down cursor keys can be used to move among the different fields of information. The left and right arrow keys move the edit cursor within a specific field. New information that is keyed in will be inserted at the edit cursor if in insert mode, or will overwrite the current character at the edit cursor if in overwrite mode.

The revision number and date fields cannot be edited by the programmer. These fields are updated automatically whenever the program is modified and saved. The revision number will increase by one and the current date will be entered.

The identification sheet can be exited by pressing the ESC key.

13.10.2. Section Title

Sections of the ladder logic program can be identified by a section title. This allows the programmer to identify a group of rungs by the function they perform. For instance, a group of 10 rungs may perform the "Robot Safety Interlock Checks". Sections are defined as all rungs located from the rung where the section title was entered to the rung before the next section title. Press the Ctrl-F3 key to enter, view, or modify the section title. The prompt "Section Title:" will be displayed in the command display area followed by the current section title, if one exists.

New text can be entered and will be inserted or overwrite the previous section title based on the insert/overwrite mode setting. If the modified title is blank, then the section will be deleted and incorporated into the previous section.

Press the ESC key when finished with the section title.

13.10.3. Comment

Comments can be associated with individual rungs. This allows the programmer to explain in detail the function of a single rung. For example, a rung in the "Robot Safety Interlock Checks" section may be described by the comment:

This rung checks that no one has stepped on the pressure sensitive mat. If someone does, the robot is emergency stopped.

Press the Ctrl-F4 key to enter, view, or modify a comment. The existing comment will appear in the command display area. The cursor keys can be used to move around the comment. New text will be entered based on the insert/overwrite mode. Press the ESC key when finished with the comment.

13.11. Help Function

This function is currently not implemented.

Pressing Alt-F1 from the main menu will invoke the Help index. Use the cursor keys to select the desired topic and press Return. The help information will be displayed. Press the ESC key when finished with the information to return to the Help index. Select another topic using the cursor keys, or select the Quit topic or press the ESC key to return to the program display.

A future revision of this software will use prepackaged subroutines that can be purchased from third party vendors for developing context-sensitive Help menus in Turbo Pascal. Context-sensitive means that the Help screens can be accessed from anywhere in the program, and the screen appropriate to the current state of the program will be automatically displayed. For

example, pressing the Help key from inside the Print function would automatically display the Print function help screen. In addition, the program would return to the Print function after the help screen was exited. This would be a major improvement over the current specification which would force the PLC programmer to exit the Print function, access the Help index, select the desired screen, and the return to the Print function.

13.12. Print Function

The commands in this section have been implemented and debugged, and are currently available for use. The Cross-reference, Address usage, and No description reports are currently not implemented.

The print function is used to print a hard copy of the ladder logic program, as well as to generate and print the cross-reference, address usage, and no description reports. These reports are useful for debugging the ladder logic program. The print function is invoked by pressing the Alt-F3 key. The print options menu (refer to Figure 13.4) will be displayed on the screen. Use the up and down cursor keys to select the different options.

Use the Up and Down cursor keys to choose a report.
Use the space bar to toggle YES / NO.
Press ESC when selection is completed.

Printer type	DOT
Ladder listing	YES
Descriptions	YES
Start rung	1
End rung	3
Cross reference report	NO
Address usage report	NO
No description report	NO

Figure 13.4. Print function menu.
Default values are shown for the reports.
Printer default value depends on startup.
End rung default is the total rung number.

13.12.1. Printer type

The printer type can be selected as either daisy wheel or dot matrix. Selecting daisy wheel causes only standard ASCII characters to be used for printing while the dot matrix setting uses the special higher-order box drawing characters available in the IBM character set. The dot matrix setting produces a higher quality ladder logic diagram, but requires greater printing time. Printing on a daisy wheel printer must always use the daisy wheel setting. However, printing on a dot matrix printer can use the daisy wheel setting for draft copies and the dot matrix setting for the final copy. The

default value for this setting is normally dot matrix, but can be changed to daisy wheel. Refer to "Program Startup" for details. To temporarily switch the printer type for the current print only, press the Space Bar until the appropriate setting, "DOT" or "DAISY" is displayed.

13.12.2. Ladder listing

The ladder listing option prints a hard copy of the ladder logic program. To select this report, press the Space Bar to make "YES" appear. The default setting for this report is "YES". Only when this report is selected for printing do the next three print menu options appear.

The descriptions option causes the ladder listing to be printed with text descriptions attached to all the instructions. This option requires a longer print time to complete. To select this option, press the Space Bar to make "YES" appear. The default setting for this option is "YES".

The start rung option allows the programmer to select the first rung to be printed. This number must be greater than zero, and no more than the total number of rungs programmed. Enter the desired start rung number, or use the default value 1.

The end rung option allows the programmer to select the last rung to be printed. This number must be greater than zero, and no more than the total number of rungs programmed. In addition, this number must be greater than the start rung. Enter the desired end rung number, or use the default value of the total number of rungs programmed.

13.12.3. Cross-reference report

The cross-reference report is a listing of each address used in the program with its assigned text description followed by a list of all the rungs that use this address. This report is useful during debugging to analyze the interdependencies among rungs.

To select this report, press the Space Bar to make "YES" appear. The default setting for this report is "NO".

13.12.4. Address usage report

The address usage report lists all addresses, marking which are currently used in the program and which are still available. This report is used to select new addresses when making modifications to the program.

To select this report, press the Space Bar to make

"YES" appear. The default setting for this report is "NO".

13.12.5. No description report

The no description report lists all addresses which do not have a text description associated with them. This report is used during final documentation cleanup to ensure that all addresses are assigned a text description.

To select this report, press the Space Bar to make "YES" appear. The default setting for this report is "NO".

Press the ESC key when the print options have been selected. A prompt will appear in the command display area notifying the programmer to turn the printer on. Press the Space Bar when the printer is ready, or press the ESC key to abort the print. Once printing has started, the screen will contain status information concerning what is currently being printed. The main screen will be restored when printing has completed.

13.13. Filer Function

The commands in this section have been implemented and debugged, and are currently available for use. The Delete current program function is currently not

implemented. The Retrieve file must be modified to first delete the current program before retrieving.

The filer function allows access to PLC program file stored on disk. Press Alt-F4 to access the filer function. The filer function menu screen displays the current default drive and directory. If no drive or directory specification is given when inputting a file name, then these defaults are assumed. These defaults can be changed within the filer function. Refer to the "Change default drive" and "Change default directory" sections for details. The filer function command menu is displayed in the command display area. Note that the function key assignments are different in the filer function than in the main menu. Each of the filer functions will be described in detail.

13.13.1. Retrieve program file

The retrieve program file function loads ladder logic program files from the disk into the computer's memory. The program can now be viewed and modified. Only files written by the PLC development system using the Save program file command can be retrieved. An attempt to retrieve non-PLC written files will generate an error. The retrieve function also checks that the file has not been corrupted in any way, for example, through a disk error.

To access the retrieve program function, press the F1 key. If a program currently exists in memory, then a prompt will ask if it is okay to erase this program from computer memory. Enter "N" to abort the retrieve and return to the existing program. Enter "Y" to continue with the retrieval. Next, a prompt will ask for the file name. To abort the retrieval at this point, do not enter a filename, but just press Return. Otherwise, enter the desired file and press Return. If the file is found and is a valid file (as discussed previously), then it will be loaded into memory.

13.13.2. Save program file

The save program file function stores ladder logic program files from the computer's memory to the disk. The program can then be retrieved later for review and modification.

To access the save program function, press the F2 key. A prompt will ask for the file name for the save. To abort the save at this point, do not enter a filename, but just press Return. Otherwise, enter the desired file and press Return. If the file is found on the disk, then a prompt will ask if this file can be overwritten. Enter "N" to abort the save and return to the existing program. Enter "Y" to initiate the save. The program will be stored on the disk as a special PLC-

written file. The ladder logic program will remain in the computer's memory during a save; the programmer does not need to retrieve the file upon completion of the save to continue editing the program.

13.13.3. Delete program file

The delete program file function deletes ladder logic program files from disk. Note that this function can delete PLC-written (using the Save function) files only.

To access the delete program function, press the F3 key. A prompt will ask for the file name to delete. To abort the delete at this point, do not enter a filename, but just press Return. Otherwise, enter the desired file and press Return. If the file is found on the disk and it is a valid PLC-written file, then a prompt will ask if the programmer is sure that this file should be deleted. Enter "N" to abort the delete. Enter "Y" to confirm the delete.

13.13.4. Clear current program

The clear current program file function clears the ladder logic program from the computer's memory. Note that this function has no effect on the program files on disk.

To access the clear current program function, press the F4 key. A prompt will ask if the programmer is sure that the ladder logic program should be cleared from memory. Enter "N" to abort or "Y" to confirm the program clear.

13.13.5. Rename program file

The rename program file function allows ladder logic program file name to be changed on disk. Note that this function can rename PLC-written (using the Save function) files only.

To access the rename program function, press the F5 key. A prompt will ask for the file name to rename. To abort the rename at this point, do not enter a filename, but just press Return. Otherwise, enter the desired file and press Return. If the file is found on the disk and it is a valid PLC-written file, then a prompt will ask for the new file name. Again, to abort the rename, do not enter a filename, but just press Return. Otherwise, enter the new file and press Return. The program file name will be renamed on disk.

13.13.6. Directory

The directory function allows for the listing of the files on a disk. The directory will list all files, not just PLC-written files, on the disk specified. The

directory function supports full path name specifications.

To access the directory function, press the F6 key. A prompt will ask for the directory to list. To list the default directory of the default drive, just press Return. Otherwise, enter the drive, pathname, and/or filename (including wildcards) specification and press Return. The directory will remain on screen until another filer function key is pressed.

13.13.7. Change default drive

The change default drive function allows the programmer to specify the drive to be used when a drive letter is not explicitly entered in a file specification. To access the change default drive function, press the F7 key. The prompt will ask for the new drive letter. Just press Return to abort the change and maintain the current default drive. Otherwise, enter a new drive letter, "A", "B", or "C", followed by Return to select a new default drive. The new default drive will be displayed at the top of the filer main menu screen.

13.13.8. Change default directory

The change default directory function allows the programmer to specify the directory to be used when a directory name is not explicitly entered in a file specification. To access the change default directory function, press the F8 key. The prompt will ask for the new directory name. Just press Return to abort the change and maintain the current default directory. Otherwise, enter a new directory name followed by Return to select a new default directory. An error will be generated if the specified directory does not exist on the default drive. Do not attempt to use the change default directory to specify a new drive. Use the change default drive function for that purpose. The new default directory name will be displayed at the top of the filer main menu screen.

13.13.9. Function not used

13.13.10. Quit

Press the F10 key to exit the filer function and return to the program main menu screen. Equivalently, the ESC key can be pressed to quit the filer function.

13.14. Simulation Function

This function is currently not implemented. Refer to the section "Long Range" under "Future Direction for the Program" for details about an alternative implementation of the simulation function. The following description is the currently planned implementation. Formal definition of the function has not been completed; the following discussion will be conceptual.

The simulation function allows the PLC programmer to debug the PLC ladder logic program by observing the effects of different input combinations on the programmed output coils. To enter the simulation function, press the Alt-F5 key.

The programmer would set up the inputs that he wants energized. He would also set up the time interval for the program scan time. The scan time will be used to update the timers in the program. By adjusting the program scan time, the programmer can emulate the real-time scan that would occur in the actual PLC. The programmer would then signal that the program should scan once. The ladder logic program would be evaluated and the outputs would be updated to reflect this new state.

The programmer can verify the results in two ways. First, he could scan through the program using the

cursor keys to examine the actual ladder logic rung by rung. Any instructions that were energized would be displayed in bold. The programmer could determine from this display exactly what conditions were causing the output coil to energize or inhibiting it from energizing. Optionally, the programmer could toggle to a global display list. This list would display all of the addresses used in the program. It would display the on/off status of the inputs, internal relays, and output coils. Timer addresses would be displayed with their accumulated times and the on/off status of the timeout bits. Likewise, the counter addresses would be displayed with their accumulated counts and the on/off status of the countout bits. This option would allow the programmer to analyze the overall performance of the program.

In addition, an option exists to allow multiple scans to occur. The programmer could enter the desired number of scans and then initiate the ladder logic program execution. After the specified number of scans occurred, the programmer could examine the results using the two options described above.

The ability to simulate the program off-line is probably the most valuable tool in the PLC development system. The cost savings associated with debugging the program before it is ever loaded to the actual PLC are

tremendous. The machine to be controlled, as well as the control equipment, is no longer tied up for extended debugging periods. Also, damage to the machine due to program errors is minimized. Indeed, an off-line simulation function is a critical part of any PLC program development system.

14. Conclusions

A number of goals were proposed in the "Thesis Description" section (Section 7). The conclusions about the feasibility of each of these goals will be summarized in this section.

The first goal was to develop a specification of the data structures and the program algorithms required by an integrated PLC program development system. A nearly-complete description was presented in the "User's Manual" section (Section 13). The simulation function that was described at a conceptual level needs to be further defined in order to complete the specification. I believe that this specification should be incorporated into any PLC development system. Other functions may be added to enhance the system, but none of the specified functions should be omitted.

The second goal was to develop the system for IBM PC-compatible computers. The program was developed specifically on a Compaq Portable computer. At this point, I still believe that the IBM PC-compatible is the proper computer system. Execution speed of the program is acceptable, and the 640K memory limitation of the PC is not a factor. The low cost and availability of PCs make them the optimal choice for a PLC program development system workstation.

Use of Turbo Pascal as the programming language for

writing the PLC development system was the third goal. The low cost of Turbo Pascal alone justifies its use. However, this low cost is coupled with more built-in features than any other PC-based Pascal compiler on the market today. The PLC development system requires a great deal of graphics display and text manipulation. Built-in Turbo Pascal functions make these operations easy. Note that the program size limitation that impeded the completion of the prototype PLC system is not imposed by Turbo Pascal, but by the 8088 processor in the PC. Fortunately, the selection of Turbo Pascal (with its built-in overlay system) will facilitate the programming to overcome this limitation.

A fourth goal was to use the Allen-Bradley programming methodology. Although this methodology resulted in more difficult programming for ladder logic program entry and display, it has two benefits. First, Allen-Bradley is still the accepted leader in U. S. market. By conforming to their standard, I believe that the PLC development system will be better accepted by today's PLC programmers. Also, this method of entry makes program storage in a linked list data structure (described in Section 10) easier to program. This factor is important because the linked list data structure is more appropriate for implementing a simulation function.

The final goal was to verify the feasibility of the PLC development system by actually programmimg a prototype. Although the PLC development system has not yet been completed, it is my belief that the work-to-date demonstrates that such an integrated system is feasible. The "path forward" (as described previously) was that the final system would be completed within 3-6 months. This conclusion should be a great delight to the many PLC programmers.

References

[1] _____, "Programmable Controller Market to Exceed $1 Billion," Machine Design, Vol. 56 (May 31, 1984), p. 4.

[2] _____, "Programmable Controllers and Computer Identity," Production Engineering, Vol. 33 (June 1986), pp. 8 (2).

[3] Allen-Bradley Company, Milwaukee, Wisconsin. FMS Roundtable, April 15, 1987.

[4] Biondi, Frank, "PLCs and PCs," Machine Design, Vol. 57 (June 6, 1985), p. 110.

[5] Crater, Ken, "Programming Automation's Future," Design News, Vol. 42 (January 6, 1986), pp. 63 (4).

[6] Curtin, Keith, "Selection and Installation Hints for Simplifying PC Troubleshooting and Maintenance Jobs," Plant Engineering, Vol. 39 (January 24, 1985), pp. 84 (3).

[7] Donovan, John W., "Programmable Controller Update: A Guide to Specifications," Plant Engineering, Vol. 39 (March 28, 1985), pp. 44 (20).

[8] Donovan, John W., "Programmable Controller Update," Plant Engineering, Vol. 40 (March 27, 1986), pp. 30 (14).

[9] Jasany, Leslie C., "Programming PCs: Is One Language No Longer Enough?" Production Engineering, Vol. 33 (April 1986), pp. 48 (6).

[10] Knuth, Donald E. Sorting and Searching, The Art of Computer Programming. Vol. 3. Reading, Massachusetts: Addison-Wesley Publishing Company, 1973.

[11] Meinhold, Ted F., "Programmable Controllers...An Overview of Sizes and Capabilities Plus Guidelines on Selection, Installation, Maintenence, and Use," Plant Engineering, Vol. 37 (November 23, 1983), pp. 52 (16).

[12] Raia, Ernest, "The Other PC Boom: Programmable Controllers," Purchasing, Vol. 101 (August 21, 1986), pp. 38 (4).

[13] Stone, J. N., "Documenting Programmable Controller Programs...With a Microcomputer," Plant Engineering, Vol. 39 (April 25, 1985), pp. 66 (4).

Appendix A. Data Structures

Pascal definition of program storage data structures
(Refer to Figure 10.1 for a diagram of the structure)

```
type cell_pointer = ^cell;

     cell          = record
                       instruction     : byte;
                       address         : byte;
                       previous_cell,
                       next_cell       : cell_pointer;
                       end;

const title_length = 60;

type title_pointer = ^title_node;
     title_string  = string [title_length];

     rung_pointer = ^rung_header;

     rung_header   = record
                       parallel_branches : byte;
                       previous_rung,
                       next_rung         : rung_pointer;
                       first_cell        : cell_pointer;
                       section_title     : title_pointer;
                       end;

     title_node    = record
                       title          : title_string;
                       creating_rung  : rung_pointer;
                       next_title     : title_pointer;
                       end;
```

Pascal definition of description tree structure
(Refer to Figure 10.2 for a diagram of the structure)

```
const descriptor_word = 8;
      description_size = 24;

type  text_description = string [description_size];

      description_pointer = ^description_table_entry;

      description_table_entry =
          record
          description_line : text_description;
          before, after    : description_pointer;
          balance_factor   : byte;
          end;

var   description_table,
      available_descriptions : description_pointer;
```

Pascal definition of the data tables structure

```
const entries_needed = 32;
      timer_counter_limit = 999;

     reference_pointer = ^references;

     references    = record
                       rung_number     : integer;
                       cell            : cell_pointer;
                       next_reference : reference_pointer;
                       end;

    in_table_entry =
         record
         eight_IO_points : byte;
         description     : array [0..7] of
                           description_pointer;
         NO_contact_reference,
         NC_contact_reference : array [0..7] of
                                reference_pointer;
         end;

    out_table_entry =
         record
         eight_IO_points : byte;
         description     : array [0..7] of
                           description_pointer;
         NO_contact_reference,
         NC_contact_reference : array [0..7] of
                                reference_pointer;
         coil_reference,
         latch_reference,
         unlatch_reference : array [0..7] of
                             reference_pointer;
         end;

    in_table = array [1..entries_needed] of
               in_table_entry;

    out_table = array [1..entries_needed] of
                out_table_entry;
```

Pascal definition of the data tables structure (cont.)

```
timer_table_entry =
     record
     accumulated_value,
     preset_value       : integer;
     description        : description_pointer;
     timer_on_reference,
     timer_off_reference,
     NO_timeout_bit_reference,
     NC_timeout_bit_reference  : reference_pointer;
     end;

timer_table_array   = array [1..entries_needed] of
                      timer_table_entry;

     counter_table_entry =
          record
          accumulated_value,
          preset_value       : integer;
          description        : description_pointer;
          counter_up_reference,
          counter_down_reference,
          reset_reference,
          NO_count_out_bit_reference,
          NC_count_out_bit_reference: reference_pointer;
          end;

     counter_table_array = array [1..entries_needed] of
                           counter_table_entry;

var  first_rung, current_rung,
     last_rung, available_rungs : rung_pointer;

     current_cell, available_cells : cell_pointer;

     input_table : in_table;
     internal_table, output_table : out_table;
     timer_table : timer_table_array;
     counter_table : counter_table_array;
     available_references : reference_pointer;
```

Appendix B. Instruction and Address Byte Coding

Instruction byte coding

Input ┤ ├	4	Branch Start	12
Input ┤/├	5	Branch End	13
Internal -( )-	16	Output -( )-	32
Internal -(L)-	17	Output -(L)-	33
Internal -(U)-	18	Output -(U)-	34
Internal ┤ ├	20	Output ┤ ├	36
Internal ┤/├	21	Output ┤/├	37
Timer -TON-	64	Counter -CTU-	128
Timer -TOF-	65	Counter -CTD-	129
Timer ┤ ├	68	Counter -CTR-	130
Timer ┤/├	69	Counter ┤ ├	132
		Counter ┤/├	133

Notes:

1. Each instruction group is defined by a specific bit in the instruction byte (bit 0 is least significant bit, bit 7 is most significant):
   - Inputs: Bit 2 is set.
   - Branches: Bit 3 is set.
   - Internals: Bit 4 is set.
   - Outputs: Bit 5 is set.
   - Timers: Bit 6 is set.
   - Counters: Bit 7 is set.
2. Bit 2 is set for all contact type (normally open, normally closed) instructions. Bit 2 is clear for all coil type (internal coils, output coils, timers, and counters) instructions.

Address byte coding

Addresses 1 - 256 are stored as 0 - 255 for inputs, outputs, and internal relays.

Addresses 1 - 32 are stored as 1 - 32 for timers and counters.

Vita

Steven M. Howard was born on June 5, 1963 in Philadelphia, Pennsylvania to Bernard M. and Carol L. Howard. He graduated with highest honors from Case Western Reserve University (Cleveland, Ohio) with a Bachelor of Science in Engineering, Major Field: Computer Engineering, in May 1985. He accepted the Kingsley Fellowship to attend the Graduate School at Lehigh University. Currently, he is employed in the Engineering Department of E. I. Du Pont de Nemours & Co., Inc. He is also a Professional Engineer-in-Training.