EVALUATION OF ALTERNATIVE APPROACHES FOR INTERACTION BETWEEN LADDER LOGIC ON A PROGRAMMABLE CONTROLLER AND ALGORITHMIC PROCESSING

by

Rajat H. Srivastava

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering and Operations Research

APPROVED:

Dr. Michael P. Deisenroth, Chairman

Dr. Marilyn S. Jones

Dr. Charles E. Nunnally

January 1988

Blacksburg, Virginia

EVALUATION OF ALTERNATIVE APPROACHES FOR INTERACTION BETWEEN LADDER LOGIC ON A PROGRAMMABLE CONTROLLER AND ALGORITHMIC PROCESSING

by

Rajat H. Srivastava

Dr. Michael P. Deisenroth, Chairman

Industrial Engineering and Operations Research

(ABSTRACT)

In this thesis an evaluation was performed of different approaches for interaction between algorithmic processing, and ladder logic and memory locations in a programmable controller. The evaluation provides a potential customer with information to make a decision for selection of the equipment used in this work. The specific capabilities of two approaches to programmable controller interfacing, a) via an intelligent I/O module and b) via a host computer on a network have been detailed. The attributes for comparison of the above two approaches were, a) capability of accessing information from memory locations in the programmable controller, b) capability of processing information in parallel to the programmable controller and c) memory considerations associated with each approach.

The evaluation was performed on a TI530 programmable controller. The intelligent I/O module used was the Texas Instruments programmable BASIC module. An IBM-PC was used as a host to the TIWAY I network. The TI530 programmable controller was configured as a secondary on the network. The BASIC module was placed on the I/O rack of the programmable controller.

The limitations and assumptions made for the purpose of evaluation, and scope for further work are presented. In addition, economic considerations and additional capabilities of both the approaches are presented.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Michael P. Deisenroth for his constant guidance and effort in seeing the completion of this work. My thanks to Dr. M. S. Jones and Dr. C. E. Nunnally for their support, and for agreeing to be on my committee. This work would not have been possible without the hardware and software donated by Texas Instruments. I also wish to acknowledge the valuable assistance provided to me by the technical staff at Texas Instruments in Johnson City.

I would also like to thank Osman Ulular, Chell Roberts and Ed De Meter for their help towards this thesis. I take this opportunity to thank N. K. Shankar and Sanjay M. Shah for their support during my stay in Blacksburg, and V. P. Mohandas for his helpful suggestions and presence.

This work is dedicated to my parents Dr. & Mrs. H. C. Srivastava, without whose blessings and constant encouragement, would not have seen fruition.

Table of Contents

List of Illustrations

1.0 Chapter 1. Introduction

1.1 Background

The National Electrical Manufacturers Association (NEMA) [14] defines the programmable 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." Bryan and Jones [10] define a programmable controller as a "solid state device used to control machine or process operation by means of a stored program and feedback from input/output devices." Definitions vary, but in simple terms a programmable controller is a general purpose microprocessor based controller which is industrially hardened and works in an I/O intensive application. It accepts inputs, evaluates them and generates outputs to control a machine or process based on a stored program.

The first programmable controllers were introduced in 1969 to replace hardwired relay control systems in industry. The operation of a programmable controller is based on a scan cycle which is continuously repeated. During each scan cycle, the central processing unit (CPU) receives input data from the system under control, performs logical decisions based on a stored program and updates outputs. Its operation therefore, is similar to scanning a series of relay contacts in real time, and its software architecture is different from main stream data processing computers.

Gradually programmable controllers were preferred over the usual relay panels and wiring, and sales for 1986 were over $71 million. The reasons for their growth in usage are: easier to use, easily reprogrammable (flexible), rugged, inexpensive and reliable, apart from having high operation speeds and easy trouble shooting. Gould [5] lists other benefits which include "reduced wiring in the field, multiple stored control programs, off line debugging, reduced change over time, immediate documentation, increased worker efficiency, faster startup, reduced spare parts inventory and varied functionability."

Initially programmable controllers did little more than ON-OFF sequencing of motors, solenoids and actuators- the principal logic functions of the relay panels they replaced. Today many large programmable controllers are commonly available with such advanced capabilities as data acquisition and storage, report generation, solution of complex mathematics, servomotor control, stepper motor control, servo axis control, self diagnosis, system trouble shooting, and communication with other programmable controllers and computers.

1.2 Operation and Components

The operation of a programmable controller is performed by scanning data continually from input devices, sequentially evaluating and updating the data based on a program stored in memory, and generating appropriate signals to control output devices. The scanning cycle of the programmable controller is comprised of four basic tasks [11]:

- I/O update (checking status of all inputs and outputs)
- Executing the logic program stored in memory in accordance with the I/O states,
- Internal execution (e.g. self diagnosis), and
- Communication with other devices such as display terminals, other programmable controllers, computers, etc.

Scans are repeated over and over again, with a per scan time from 2 to as much as 200 milliseconds (depending upon the particular programmable controller, size and structure of the program, and number of I/Os). Inputs are signals from sensors in the control system, and outputs are signals that actuate the devices/actuators to control a machine or process.

The main components of a programmable controller are it's central processing unit (CPU), memory, I/O modules and power supply.

1.3 Comparison with a computer

A computer executes each step of a stored program in a minute fraction of a second, but the complete program can take several minutes or longer to complete, depending upon the program. Such an operation would be out of question for a typical programmable controller, the primary function of which is controlling the operations of processes and machines via simple logic decisions. For the programmable controller to continuously and virtually simultaneously analyze process input conditions, make logical decisions, and drive outputs, the complete program needs to be executed in a small fraction of a second [1]. Since the processor on the programmable controller is no faster than those on conventional computers, the program is normally shorter and simpler. The programmable controller is designed specifically for the manufacturing environment and is not a general purpose data processing machine.

In general purpose computers, tasks can be executed in any order. Figure 1 on page 5 [11] illustrates the program structure for a programmable controller and for a general purpose computer. A programmable controller is slower than a general purpose computer because of its software structure, which needs sequential execution. It does not allow concurrent execution of tasks or the expenditure of more than the minimal time on any one task. This leads to a certain amount of inflexibility. A conventional computer is a general purpose device in that it is usually used to edit, compile and run a number of different programs one after another. The programmable controller on the other hand, is a special purpose device and typically executes its small program continuously several million times before a new program is introduced.

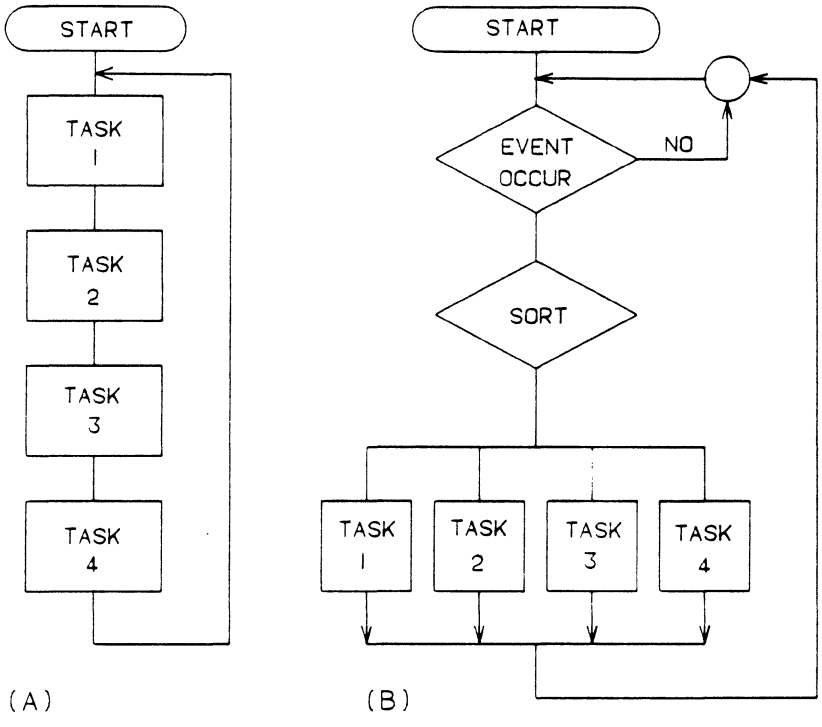


Figure 1. Program Structure for programmable controllers (A) and computers (B).

Another major distinction between programmable controllers and computers is in the use of programming languages. For programmable controllers, they are designed to deal with many I/O points which are part of the control system. For computers, languages are oriented towards scientific or business data processing. Lately however, some flexibility has been added to the larger programmable controllers by including instructions that allow subroutine calling, interrupt routines, and a means for bypassing or jumping certain instructions. There are currently four major languages widely used with todays programable controllers [11]: boolean equations, logic diagrams, mnemonic programming and ladder logic. For the purpose of this thesis, programming of the programmable controller was done in ladder logic.

Another area where programmable controllers have needed improvement is in operator interfacing. Better displays, giving improved diagnostics, and sophisticated programming languages, are currently being offered by many vendors. However, there is still a long way to go before programmable controllers give satisfactory status reporting, reports and graphic displays.

Initially the use of a personal computer for control applications was only useful if the I/O requirements were limited, the user could program in a high level language and the environment was airconditioned and clean. However, rugged industrial computers (personal computers)-without fans, filters and disk-drives, increased I/O capabilities and battery backed random access memory (RAM) are now being made available.

Therefore, it can be seen that the differences between programmable controllers and computers is narrowing in most respects. The basic difference is primarily one of software architecture. Typically, users want to separate control functions into the

programmable controllers with its ruggedized design, while using computers for computational tasks, and viewing and reporting functions.

To avoid confusion in this document, the programmable controller will be designated as a P/C, and the personal computer as a PC.

1.4 Interaction between Algorithmic Processing and Ladder Logic

As factories automate to survive, there is need for real time information to reach all departments of the factory, especially supervisors and managers. Several P/Cs may be connected to a network under the supervision of a computer, which accesses information from the P/Cs and controls their operation based upon its own stored program.

P/Cs perform a variety of functions and their capabilities can be vastly improved with intelligent I/O modules, which have their own microprocessors and which run asynchronously of the P/C. These intelligent I/O modules can store programs in memory and can communicate with and control the logic program in the P/C.

The above functions entail an interaction between algorithmic processing and ladder logic. The interaction is possible by one of three methods:

- A computer program stored in an intelligent I/O module of the P/C can communicate with the P/C,
- A program in a host computer can communicate with the P/C via a network for distributed control, and
- A program in a computer can communicate with a P/C directly through the RS-232 port on the P/C.

1.4.1 Intelligent I/O Modules

Most manufacturers of P/Cs provide input and output points in modular form (I/O modules with status indicators) on I/O bases, for flexibility, ease of maintenance, control over a large application area and reduced cost of system wiring. The I/O modules for a P/C serves several functions [11]. Their main function is to reduce the input voltages to logic signal levels which can be understood by the CPU, and to convert the logic signal levels (outputs in the control program) to voltages, so that connected devices (actuators, for example) can be controlled. They also protect the CPU from electrical noise and provide interfacing for devices such as bar code readers. Input devices could be any of limit switches, strain gauges, thumbwheels, pushbuttons, thermocouples etc.

A variety of I/O modules exist for different applications. Some of them are: discrete AC/DC output/input, analog input/output, word input/output and special function modules. Some of the special function modules are: ASCII I/O, Stepper Motor (pulse) output, Axis Positioning, PID (proportional, integral, derivative control), BASIC and NIM (network interface module). Some of these special function modules are intelli-

gent, i.e. they have their own microprocessors which are either preprogrammed by the manufacturer or can be programmed by the user.

Some of the functions performed by these special function modules, eg., PID, can be handled by a subroutine in the P/C with more basic I/O modules. This was the procedure commonly used for PID control before intelligent I/O modules were introduced. However, the advantage of using the intelligent I/O modules is that whatever is being processed in them, is handled at microprocessor operating speeds. Processing within the module is done in parallel with the processing within the P/C, and this relieves the program memory requirement and processing time on the P/C processor. It also enormously simplifies the task of creating the control program (in ladder logic). Even a simple bubble sort program can be fairly tedious in ladder logic.

The BASIC module [21] is an intelligent I/O module which can be programmed in the BASIC programming language. It is used mainly in applications where:

- Complex mathematical calculations (as in PID, statistical analyses, and gas flow calculations) are needed,
- Machine diagnostics are required,
- Parallel processing is needed to shorten the P/C scan time, and
- Operation with specialized devices such as bar code readers and CRTs is desired.

With the BASIC module, information is exchanged between the module and the P/C via subroutine calls in the BASIC instruction set. Communication between the programmable controller and BASIC module requires interaction between algorithmic processing and ladder logic.

1.4.2 Distributed Control

Figure 2 on page 11 [21] illustrates a typical partitioning of control in a modern factory. Since factory host computers handle order entry, factory scheduling and material requirement planning in large companies, it is imperative that data from P/Cs reach the factory host computer. In addition, with P/Cs taking on more and more sophisticated tasks, there is constant pressure/need for faster scan times and greater computing power on the shop floor. Distributed control meets this need by breaking down the overall control task into smaller functions. Independent controllers are used for each function, thereby speeding execution and reliability. This is possible due to reduction in cost and size of microprocessors, improvement in speed and reliability of data communications, and the increased sophistication of software which enables multiple computing devices to function harmoniously in a network.

There are two viable methods for achieving distributed control at the machine/process level. One is by a high speed, high performance Local Area Network (LAN), wherein computers can communicate with P/Cs. Since this method of communication involves interaction between ladder logic and algorithmic processing, it is being considered for this research. The other means is by direct P/C to P/C connection, wherein the outputs of one P/C are accepted as inputs to the other, and thereby information about a process can be transferred. Since there is no interaction between algorithmic processing and ladder logic by this approach, this matter will not be treated further.

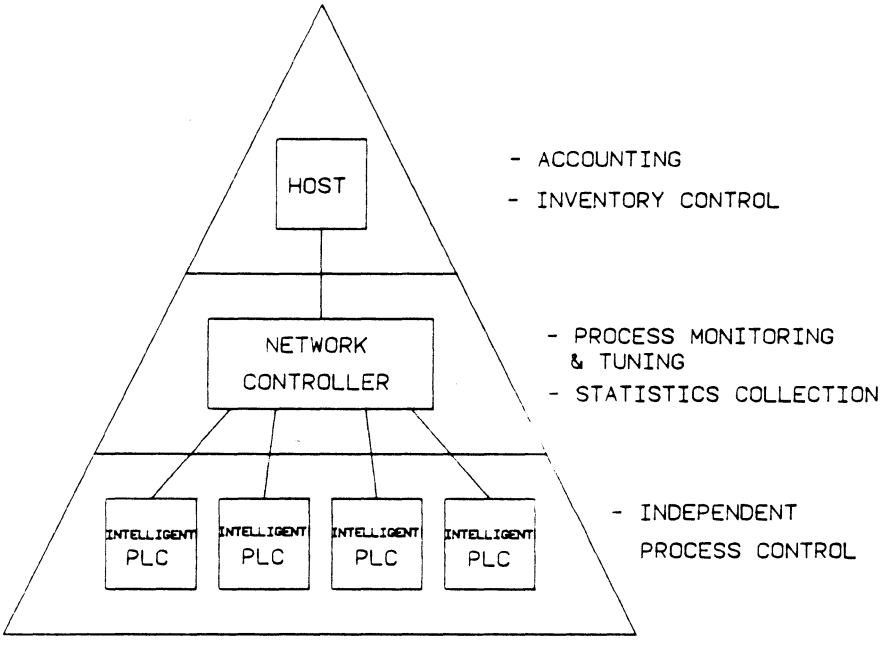


Figure 2. Distributed Industrial Control

1.4.3 Direct PC-P/C Communication

The RS-232 communication port on the P/C is typically used for programming of the P/C. However, it can also be used for communication with other devices, such as printers, other P/Cs and computers. Information can be exchanged between a PC and the P/C directly through the RS-232 port, by means of task codes. This manner of communication is not going to be treated in this research for reasons detailed in the following section.

1.5 Objective

The objective of this thesis was to perform tests to evaluate two of the above three methods of interaction between ladder logic on a P/C and algorithmic processing. Specifically addressed were, interaction via 1) an intelligent I/O module, and 2) via a PC as host on a network with the P/C as a secondary device. The criteria that were used for testing were data transfer rate capabilities under two typical industrial applications. The first application involved I/O intensive tasks, wherein information from several data points needed to be transferred between the P/C and the attached intelligent device, to perform the control function. The second application was one which involved computationally intensive tasks. Here, the attached intelligent device performed a large data processing task in parallel with the processing on the P/C. The value or values (which were the result of processing of the data) needed to be returned to the P/C within a predefined time, depending on the process under control. For each of the tests conducted for computationally intensive tasks, the size of the

tasks on the BASIC module were noted, to establish the size of application programs for which available memory on the BASIC module became a limitation. A discussion is also presented concerning the economics of each alternative, and other rated capabilities not under the scope of the above tests.

The system under control of the P/C was a working physical model of a flexible manufacturing system. The model consisted of a parts loader, a conveyer, a machining station having drilling, milling and laser welding facilities and a sorting station for finished goods. For the set up in the first approach, the LAN that was used was the TIWAY I network with an IBM-PC as the host. The TI530 P/C acts as a secondary on the network under control of the host computer. For the second approach, the BASIC module was mounted on the I/O base of the P/C, and communicated with the P/C via the bus on the I/O base. Programming of the P/C and the BASIC module was done on a VPU200 (a factory floor programming and troubleshooting device).

Communication by direct connection via the RS-232 interface was not considered for the following reasons:

- The amount of information that can be obtained per query by this method is limited.
- Maximum baud rate between the PC and the P/C is limited to 9600.
- This approach is not commonly used for control applications.
- Program development by this method is more difficult because communication has to be done using task codes in assembly language.

2.0 Chapter 2. Literature Review

2.1 *Introduction*

Most of the over 60 manufacturers of P/Cs have products which allow communication between algorithmic processing and ladder logic, particularly by way of networks and intelligent I/O modules. P/Cs have a specialized type of LAN called 'data highway' [12]. They are optimized for their specialized shop floor duties, and are relatively inexpensive. Major manufacturers of P/Cs and their LANs include the following:

Allen Bradley: Data Highway

Gould Modicon: MODBUS

Texas Instruments: TIWAY I

Honeywell: CIM

Square D: SY/NET

Most of the developmental work done for communication between algorithmic processing and ladder logic is proprietary in nature, and therefore difficult to obtain. The same is true for most of the applications by users in industry. However, in the past

couple of years there have been a few application oriented publications in this area. The literature review describes some of those applications where P/Cs have been interfaced with PCs. Some alternative approaches to PC-P/C interfacing are also presented.

2.1.1 Design steps for a monitoring/control system

John M. McLoughlin of Price Waterhouse [13] has outlined a systematic approach for structuring, organizing and presenting data, and structuring software for a monitoring/control system. It is an aid to designers who wish to use microcomputers in the factory with particular reference to collection, manipulation, and presentation of information from P/Cs. The steps in the design process are as follows:

a) Define the type of data which will be communicated between the P/C and the microcomputer. Data can be passed in both directions. Typically one or more of three data classifications needs to be communicated: production, process or diagnostic data.

b) Define the arrangement and placement of retrieved data in the P/C. Careful design of placement of data can help simplify the microcomputer software.

c) Determine the software requirements. The software must be capable of accepting raw data values from a field device, converting it into information, and recognizing the status of those values. Software requirements are met by driver packages which are used for communication with particular brands of P/Cs, and for display and reporting capabilities.

d) Obtain the communication hardware, which can be any commercially available LAN. These systems allow for direct register and word address reading and writing, thus freeing the P/C CPU from the task of creating communication messages.

e) Decide on the microcomputer. Typically, requirements are speed, power, familiarity and availability of software.

2.2 Applications using PC-P/C communication

G. D. Hill and R. H. Deane [7] of Pentek Corp. and Georgia State University respectively, have integrated a PC and a P/C to manage material flow and in-process storage facilities in a manufacturing plant. Controls are needed at the location of all in-process job orders on temporary storage conveyers, since manual tracking of in-process orders is difficult due to the jobs being fairly indistinguishable at that stage of production.

An Allen Bradley Model 2/30P/C and an IBM-PC perform the necessary decision making and control in the system. The IBM-PC serves as the coordinator of the system by maintaining information on all jobs processed during the day. Information about the line-up of jobs can be either manually entered, or downloaded from a mainframe computer, or transferred via floppy disks to the PC. The Allen Bradley Communication Adapter Module (Model 1771-KA) allows communication between the P/C and the IBM-PC via a proprietary Allen Bradley 'data highway' system. An Allen Bradley stand-alone communications controller module (Model 1771-KF) serves as

an interface for the IBM-PC. This total configuration allows communication between the IBM-PC and the P/C.

The Allen-Bradley P/C is the interface control to the material handling equipment. Based on the schedule previously determined, the IBM-PC on the other hand, determines the storage location of the job, the existing distribution of jobs, and the particular machine to which each job has to go for the finishing operation. In this application a PC has been dedicated for use with a single P/C.

Robert Hagen of Koester Corporation [6] has interfaced a Modicon 484 to an IBM-PC. The main requirement for the interfacing is to increase data handling capability and for ease of operator interface, using a menu driven system. The need for using a PC for this requirement, is for two reasons. Such features are available in large P/Cs, but the complexity of data handling statements in those P/Cs defeats the original intent of their ladder logic programming by plant electricians. The second reason is that PCs are available at fairly low costs as compared with large P/Cs.

The system controlled by the P/C is a robot and 3 conveyers full of various parts. The PC downloads a specific instruction set to the P/C for control of the robot, for a particular product mix. The capabilities of the total system are:

- Graphical display of current inventory of parts in the system,
- Production history storage,
- Temperature and humidity displayed and stored in history,
- Alarm messages history and maintenance work history, and
- Downloading history to floppy disks for use on LOTUS 1-2-3.

The communication link between the PC and the P/C is direct via an RS-232C interface at 9600 baud. The communication software which was developed, determines speed, versatility and error checking of data that is passed between the two devices. The IBM computer is the master, and the P/C is the slave. The P/C transmits data only when requested. When the P/C performs an operation that changes the status of inventory, it waits till the PC acknowledges that it detected the change and updated its inventory accordingly. This handshake is necessary in case the PC is being used for some other function at the time. In this application, PC-P/C communication has been done without networks and intelligent I/O modules.

James J. Pinto of Action Instruments Inc. [17] considers PC to P/C communication through multiprocessers. The shortcomings of P/Cs, which are limited data analysis capability and lack of convenient operator interface, were the motivation for the PC to P/C communication. However, the problem that he has addressed is that of the PC being unable to cope with high speed P/C communication due to it's (PCs) single processor architecture. While performing other ordinary application tasks, PC-P/C communication can slow down to an unacceptable level. Even with multitasking, the communication with several P/Cs becomes impractical. Multiprocessers are used to overcome this problem with separate co-processors taking the real-time communication burden and relieving the host processor (PC) to supervisory tasks, database management and operator interface.

The real time communication coprocessor system that has been developed, allows four separate and different P/C protocols to communicate with a standard PC (IBM-PC/XT/AT). It is the SECOM-4 communication coprocessor subsystem for mul-

tiple P/C communication with the IBM-PC. The communication coprocessor hardware module has four RS-232 communication channels, all of which may be run simultaneously, at high data rates. Essentially, communication between P/Cs and the PC is through these RS-232 ports.

Conventional distributed control systems have emerged as technology for continuous regulating, monitoring and supervising. Large control and distributed control systems have included P/Cs interfaced by various methods. Harris A. Carr of Leeds and Northrup Co., North Wales, PA [2] has developed a process control system which offers built-in P/C capability, thereby avoiding interfaces altogether. The features of such a system need to integrate digital P/C type operations and analog distributed control system type continuous functions. An advantage of placing the P/C activity into the distributed system is that it also provides a good operator interface.

The controller developed above has multiple microprocessors. One each is used for the following functions:

- Input processing,
- Executing control algorithms,
- Communicating over the data highway,
- Performing user defined functions, and
- Communicating through RS-232, RS-422 and RS-485 serial links to external equipment.

Using multiple microprocessors and co-processing techniques, the controller provides powerful processing capability to process inputs, perform outputs, perform re-

quired algorithms and calculations, and generate outputs twice each second, since all functions are performed in parallel without interfacing with one another. Such distributed controllers can also be connected via a network, and are in use in a variety of applications.

Shirley B. Gosset [4] has designed and implemented a system that can handle communications between a host computer and P/Cs for a single flexible manufacturing line, at AT&T Technology Systems, Inc. Richmond, Virginia. Available commercial software was not able to connect the host computer to the P/Cs, because of the host computer that was being used. The communications handler which was developed, is a communication protocol converter between P/Cs and the host computer. It also acts as an operator interface and can be used as a system controller on a P/C LAN. The following are the capabilities of the system controller:

- Send/receive data to host computer via a standard LAN,
- Send/receive data to P/C,
- Read/write data to a disk,
- Perform data calculation, and
- Move blocks of data from one P/C to another or to different variable types within a single P/C. One of the requirements of the controller was to be able to allow peer-to-peer communication between P/Cs.

A part of the system is a man/machine interface (MMI). The MMI is used for gathering data from the P/Cs on the P/C network (MODBUS LAN) and displaying them in an

attractive and easy-to-read manner. The MMI is an AT&T 6300 computer and runs on either Microtie or Indelec (commercially available packages).

The above capabilities of the communications handler were predefined system requirements. Other performance criteria which were considered during design of the final system were: a) ease of operation and maintainability, b) cost and c) flexibility. The predefined system requirements led to an evaluation of available alternatives in the design of the communication handler. The following three alternatives were available:

a) The communications handler is placed between the host computer and the MMI, which is connected to the P/Cs.

b) The communications handler is attached to the P/C (MODBUS) network and appears like a P/C to the MMI. The communications handler is directly connected to the host computer but cannot directly talk to P/Cs and has to be a slave to the MMI.

c) A second MODBUS network is created with the communications handler as its master. The MMI is the master of the first network. A second communication port is added to each P/C for the second MODBUS network.

In each of the alternatives the host computer communicates with the system controller via an RS-232 serial port at 9600 baud. The third approach was selected because it was the only one which allowed peer-to-peer communication, and satisfied all other operational requirements.

2.3 Summary

The literature review presents the available published industrial applications in the field, considering all the methods available for interfacing computers and P/Cs. It is observed that a large number of companies are interfacing P/Cs to be able to access shop floor information for upward integration and for simplifying the decision making process. Two important reasons for interfacing the P/C with the PC are evident in the common limitations of the P/C- namely, data processing and operator interfacing. As factories become more and more sophisticated, there is need for more computing power at the P/C level, as well as enhanced feedback and reporting functions. A lot of applications are dedicating a PC with a P/C for these reasons.

Three principal methods for interfacing P/Cs, which involved interaction between algorithmic processing and ladder logic have been reviewed:

- Via a network (LAN),
- Via an intelligent I/O module, and
- Direct connection via the RS-232 interface on the P/C.

Presently there is no literature on comparison and evaluation of methods for interfacing P/Cs, which involve interaction between algorithmic processing and ladder logic.

3.0 Chapter 3. Methodology

3.1 Introduction

An evaluation was performed of two approaches used for interfacing ladder logic with algorithmic processing : a) via an intelligent I/O module, and b) via a PC as a host on the network with the P/C as a secondary. The evaluation was performed by testing both the approaches under criteria typically used in an industrial environment. The factors which influence P/C interfacing are: a) speed with which data can be processed in the interfaced device, and the response sent back to the P/C, b) rate at which data can be transferred between the P/C and the interfacing device, and c) available memory on the interfacing device.

3.2 Test system configuration

The system under control of the P/C was a working physical model of a flexible manufacturing system. The IBM-PC was connected to the P/C via a TIWAY I LAN, with the PC as the host on the network. The TI530 P/C acted as a secondary on the network under control of the host computer. The BASIC module was mounted on the I/O base of the P/C, and communicated with the P/C via the bus on the I/O base.

3.2.1 Network

Figure 3 on page 25 [20] illustrates the LAN configuration which was used in the research. The basic components of the network configuration were an IBM-PC, a network host adapter, and network interface module. An IBM-PC was connected to the network via an RS-232C serial port on an asynchronous communications card. The IBM-PC was connected to the network host adapter, which was tied to the NIM (network interface module). Since there was only one P/C on the network, the NIM was directly wired to the host adapter. The NIM occupies a slot on the I/O base of the P/C and communicates with the P/C via the bus on the I/O base.

There are different levels of communication between each of the network components. Figure 4 on page 26 [8] illustrates the different levels of communication. At the most basic level, there is a task code communication between the NIM and the P/C. Each model of P/C uses a different set of task codes. The next level of communication is between the Host Adapter and the NIM, in which primitives (or the

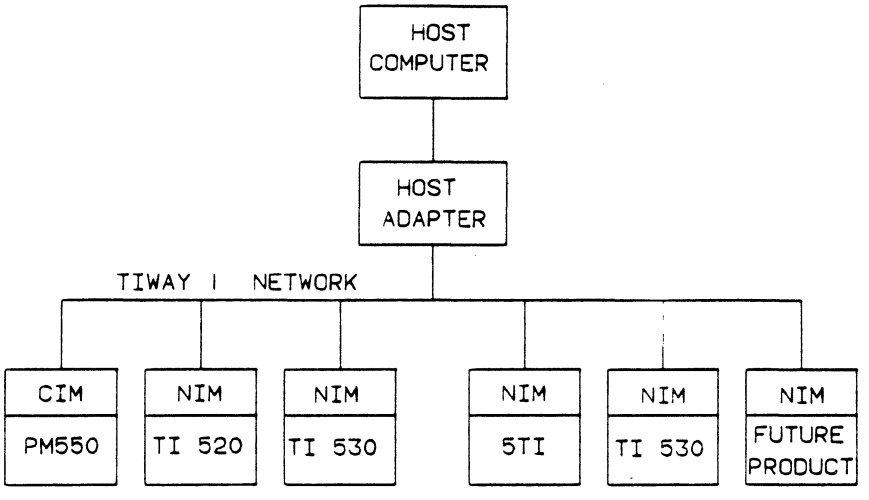


Figure 3. TIWAY I network configuration

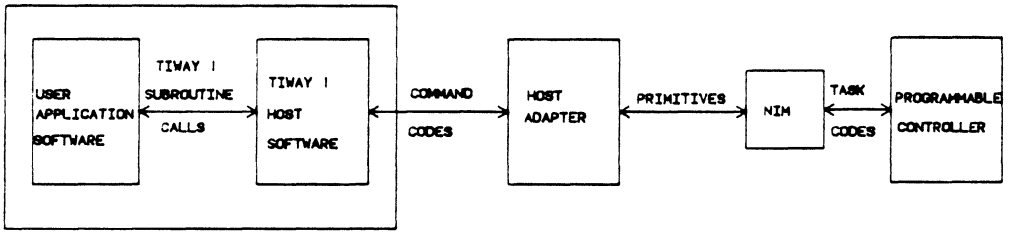


Figure 4. Network Communication Levels

universal command language instructions) are used. The universal command language (UCL) is a language that is recognized by all NIMs. Communication between the Host Adapter and the host computer is accomplished through the use of command codes. At the highest level, there is communication between the programmer and the host computer. This level involves the host software, which is a library of subroutines that need to be called to be able to communicate with the P/C, and is used to generate the command codes. The baud rate between the PC and the host adapter was 9.6K, and that between the host adapter and the NIM was 115.2K.

With the host software, the PC can access information from the P/C, perform data processing and write to memory locations in the P/C. Information which is accessed can be archived and subsequently used to format a graphical display with appropriate software. Before writing to the P/C, an operator can be interactively queried for appropriate control data values. The PC can also format reports based on the information collected and archived.

3.2.2 BASIC module

The BASIC module is mounted on the I/O base of the P/C and communicates with the P/C via the bus on the I/O base. It has 28K bytes of available memory and works asynchronously of the P/C. The BASIC module is programmed in BASIC and has five interface subroutines in its instruction set, that obtain information from, or send information to, the P/C. They are: IOREAD, IOWRITE, PCREAD, PCWRITE and SRTC (send and receive task codes). Due to limited available memory, the BASIC module cannot format reports or graphical displays. The BASIC module (like the TI530 P/C)

can retain only one program in memory at a time. There is no facility available for external mass storage. The module was programmed via the same video programming unit (VPU 200) that was used to program the P/C itself. Every program has to be stored on a separate floppy disc, and the memory of the module has to be cleared before a new program can be loaded onto the module.

3.2.3 Control system

The system under control was a physical model of a flexible manufacturing system (FMS), and is illustrated in Figure 5 on page 29. Three different product types can be loaded onto the conveyer by the three parts loaders. Parts move down the conveyer till they reach a diverter. Requests for several parts to be machined can be queued. The diverter transfers parts from the conveyer to a turn-table which indexes to reach the first machining center, which is drilling. Once drilling is completed, the table indexes again to reach the milling station and finally to the laser welding station. After all three operations are completed, the parts diverter transfers the part again onto the conveyer. The part travels on the conveyer till it reaches the sorting station, where each part is selectively pushed down a specific chute. The system was completely controlled by the ladder logic program stored in the P/C.

3.2.4 Data transfer capabilities

Data transfer capability can be defined in terms of the ability of a device to read from or write to memory locations in the P/C. During each scan of the P/C there is a cer-

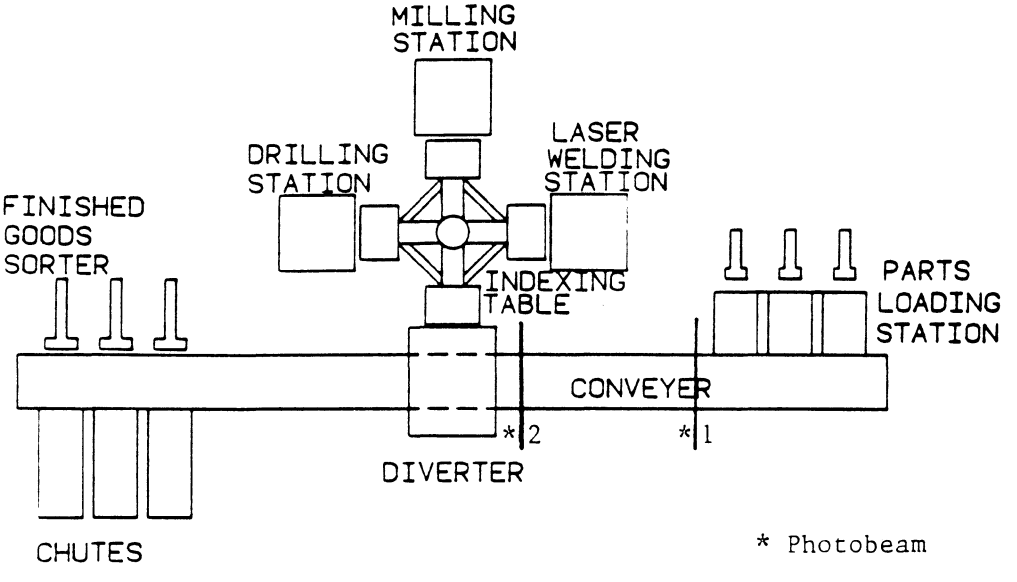


Figure 5. FMS model

tain time slot allocated for communication with any attached intelligent (special function) device. This is true only if the device is active, i.e. requesting information. The same is true if an output is to be interactively displayed on the VPU200. Each scan of the TI530 P/C takes about 8 msec to process 1K of ladder memory. Additionally, if an intelligent device were to communicate with it, the scan time would typically increase to 13 msec.

The TI530 CPU memory is divided into two parts, a) that which the user cannot directly control (operating system) and b) that over which the user has direct control (user memory). User memory consists of image registers, presets, current, status word, ladder memory and user variable memory. Some of the image registers are

a) Discrete image registers which provide single bit storage location for the status of each discrete input (X) and discrete output (Y),

b) Word image registers which provide 16 bit storage location for the value of each word input (WX), and word output (WY), and

c) Control relay image registers which provide a single bit storage location for the status of each control relay (C).

PRESET values are drum and timer/counter memory locations where preset times and counts are stored as 16-bit words, eg.,

DSP: Drum step preset, and

TCP: Timer/Counter preset

CURRENT values are drum and timer/counter memory locations where the current count and time values are stored as 16-bit words, eg.,

TCC: Timer/Counter current, and

DSC: Drum/Event step current

Status words provide the information for the CPU, I/O modules and I/O bases. The user variable (V) memory is a word memory location used as storage area for values for math instructions, drum step counts etc.

In this thesis, discrete data points refer to discrete image registers (X & Y) and the control relay image registers (C). Word data points refer to the remaining user memory areas (except ladder logic storage area).

Both of the approaches to P/C interfacing under consideration have specific capabilities as far as data transfer is concerned. The BASIC module can process only 26 consecutive discrete data points per scan of the P/C, and can read a maximum of 30 consecutive word locations per scan of the P/C. For any number of consecutive data points that need to be accessed (word and discrete), only one call statement is required for the BASIC module. If more information is required than can be processed in one scan of the P/C, it is obtained in subsequent scans. The call is completed only after the entire information has been transferred.

The host computer (with the TIWAY I software) can read a maximum of 134 consecutive words per read statement at 28 words/scan. To read more than 134 consecutive words, additional read calls need to be made. Twenty eight words is equivalent to 28*16 bits = 448 bits/scan. Discrete data points X, Y or C can be read by the host in the compressed format, i.e. one word representing the state of sixteen discrete data points. The host computer via the network, i.e. through a read primitive in the NIM, can process two task codes at a time to read 448 consecutive discrete data points per scan of the P/C.

In typical industrial applications it might be required to access different types (words and discrete) of data points that are non-consecutive. In the BASIC module, with each PCREAD statement two different discrete or word data points cannot be read, i.e. information from X12 and C101 cannot be accessed through the same read statement. By using the network, information from 32 different (non-consecutive) discrete and word data points can be accessed via one GATHER subroutine call. With appropriate ladder logic software design (i.e. by assigning consecutive C points to distinct X, Y or C points), the need for accessing up to 26 non-consecutive discrete data points can be met by the BASIC module, through a single read statement. However, separate read statements are needed to access different non-consecutive word data points, or a combination of discrete and word data points.

Response from the network is slow because information has to flow at 9600 bps to and from the PC to the host adapter, whereas that from the BASIC module is accomplished at bus speeds since it sits on the I/O bus.

3.2.5 Tests for data transfer capabilities

A test was performed to determine the time taken for a PCREAD statement in the BASIC module (to access a word data point in the P/C), for a read command over the network, and for a GATHER subroutine call on the network. The results are tabulated and presented in the form of a graph for a combination of a number of different, non-consecutive word and discrete data points locations that were read.

A second test was performed to determine the time taken to read consecutive data points (both word and discrete in the P/C), by the read commands in the BASIC

module and on the host computer via the network. These results are tabulated as well as presented in the form of a graph, with a stepwise increase in the number of data points accessed.

3.2.6 Tests for computationally intensive tasks

Keeping the number of data points to be read to a minimum, two application oriented tasks were performed which had a large computational component. The first application was one which involved linear programming, and the second involved queueing and scheduling. The objective of this test was to evaluate the two approaches to P/C interfacing on the basis of response from the attached intelligent device, in applications which involved heavy computation running parallel to the P/C.

3.2.6.1 Linear programming

The first computationally intensive task was performed by running a linear programming (LP) problem. The same algorithm was used for both the approaches so that the results would be independent of the type of algorithm used. The solution of the LP problem is based on the revised simplex method. The size of the LP problem was increased step-wise (for variables and constraints), till the memory on the BASIC module was full. The input to the LP software is the number of variables and constraints. The software generates the required constants for the problem.

The time taken for the response from the BASIC module and from the network was measured by an external timer connected as an output to the P/C. A graph of number

of variables and constraints against the corresponding response time for a successive increase in number of variables and constraints, was plotted for both approaches. On the response time scale there are set points present which represent time intervals for distinct events on the physical system (FMS model).

3.2.6.2 Queueing and scheduling

The physical model already consists of three parts loaders, and it is feasible to imagine three queues being formed in front of the three part loaders, with ten jobs per queue. In terms of software, the number of queues can be increased to a large extent, with parts from those queues being loaded onto the conveyer by one of the three physical queues.

The time taken for the response from the BASIC module and from the host computer on the network, was measured by the same external timer connected as an output to the P/C. The number of queues was increased stepwise till the memory in the BASIC module was full. A graph of number of queues (with 10 jobs per queue) against the corresponding response time was plotted for both the approaches to P/C interfacing. As in the linear programming task, set-points which represent time intervals for distinct events on the physical system (FMS model) are present on the response scale.

FORTRAN was used to generate the test programs for the PC using the TI/IBM host software for the IBM PC, and interpreted BASIC (the only one allowed) to program the BASIC module. For the linear programming and the queueing and scheduling of computationally intensive tasks, the programs were also written and executed in

compiled QUICKBASIC and interpreted BASIC languages over the network, to compare operation of the network based on different programming languages.

3.3 System hardware

The hardware used in this research consists of the following five major components: the TI530 programmable controller, the video programming unit (VPU200), the IBM-PC, the BASIC module, and the TIWAY I network.

The TI530 [16] is a midsize programmable controller from Texas Instruments. It offers standard local and distributed I/O. The I/O bases may be located up to 300 meters from the CPU. Each I/O module has 8 discreet I/O points, which allows for high density installation. It can address a total of 1023 I/O points. The scan time of the P/C is 8 milliseconds per 1K of logic memory.

The VPU200 [16] made by Texas Instruments, is a basic factory floor programming and troubleshooting tool for the TI530. The VPU200/530 software package has to be used to be able to program the TI530. It can be connected to the P/C directly or via a modem. The auxiliary port allows for connection to several types of printers.

The IBM-PC had 640K RAM and a 10 Meg hard disk. It also had a math coprocessor installed. Two asynchronous communication cards were installed. One was used for communication with the TIWAY I network, and the other for a serial printer. The TIWAY host software (TI/IBM PC Version 1.2), including the device driver for communication with the network was installed in the PC.

The BASIC module [19] is an intelligent I/O module which occupies any two slots on the I/O base of the TI530. It has two serial ports with either an RS-232/423 or 20ma current loop interface, 28K bytes of memory, and works asynchronously with the P/C. Basically, it is an industrial-hardened microcomputer. Programming of the BASIC module was also performed via the VPU200. The VPU200 was booted up with the Operating System software BASIC/500 (from Texas Instruments).

TIWAY I [22] is a local area network (LAN), which is designed for industrial environments. It connects a series of TI P/Cs with a host computer. By proper use of the network, and appropriate software, one can obtain, modify, or replace data stored in separate program memories of a network of interconnected P/Cs. TIWAY I supports the bus topology using a primary/secondary scheme that is easy to implement, and has distinct advantages in networks using P/Cs as secondaries. It uses the polling scheme for network access. TIWAY I is a hosted network; a primary (host) computer controls up to 254 separate secondaries, providing a central collection point for information. TIWAY I uses the industry standard High Level Data Link (HDLC) protocol, and confirms to CCITT X.25 standard for packet switching. The Host Adapter [20] provides the means for connecting a host computer to the TIWAY I communications network, allowing communication with up to 254 P/Cs on the network. It supports data rates from 110 to 19.2K bps via an RS-232C serial interface for communication with the PC. It uses TIWAY I network instructions (Primitives) for communications between the host computer and the network interface module. The Network Interface Module (NIM) [21] is an intelligent module which occupies one slot on the I/O base of the network secondary (P/C) to which it is attached. Each fully conformant secondary on TIWAY I has a NIM installed. The NIM functions as an HDLC secondary on the TIWAY I network.

3.4 System software

Several software system components were needed for this research.

The VPU200/530 operating system software (rel:3.0), is used for development, execution and debugging of ladder diagrams on the VPU200. One can check the status of the system under control at any time interactively, either in ladder or tabular format. Programs can be downloaded to a floppy disk from the P/C or uploaded to the P/C from a floppy disk.

The VPU operating system software (Rel:1.0, BASIC/500), enables programming and execution of programs on the VPU in the BASIC programming language. Programs may be downloaded from the memory of the BASIC module onto floppy disk or uploaded from floppy disk to the module.

The TIWAY host software (TI/IBM PC Version, Rel:1.2) is the interface between a host computer and the TIWAY I communication network. It provides access to all the secondary functions available through the network, as well as a complete set of network management and diagnostic facilities. It makes all the network commands (primitives, task codes etc.) transparent to the user and allows programming in high-level languages (BASIC, FORTRAN, PASCAL & C).

Other software used were, a) DOS version 3.0, b) MS Linker version 3.06, c) MS FORTRAN version 3.31 and d) MS QUICKBASIC version 2.0.

4.0 Chapter 4. Development of tests for evaluation

4.1 Introduction

In this chapter each of the programs and subroutines developed and used for the tests are discussed in detail. The programs are grouped under two primary categories, 1) test programs for computationally intensive tasks and 2) test programs for data transfer capability. The interface command subroutine calls made to the P/C during each of the programs are detailed, as are the relay ladder logic (RLL) programs running in the P/C at that time.

4.2 Set-Points

Initially set points with respect to time were established with reference to the physical model of the FMS, for both the computationally intensive tasks. These set points

provide a comparison of the different approaches to interfacing P/Cs with respect to the system under control. These set points are incorporated in the response time graph (which is the result of this test). With respect to these set points which correspond to the FMS model, it can be known what size and type of computation is possible before a certain event occurs (for the different approaches to P/C interfacing under consideration). Three set points were established for the queueing and scheduling computationally intensive task, and two for the linear programming task.

To define the set points a program was written for the P/C in relay ladder logic (RLL), to control a portion of the physical system. This program starts the conveyer in the FMS model. Upon request by an operator, a block representing a part is loaded from the first parts loader onto the conveyer. The conveyer carries the part till it reaches the diverter, whereupon the conveyer stops. The diverter moves the part from the stationary conveyer onto the indexing table. When the diverter is fully extended, the indexing table indexes 90 degrees till the part on the table reaches the drilling station. Henceforth, the part will be designated as a job.

An external timer capable of measuring up to 1/100ths of a second was connected as an output to the P/C. The start and stop of the timer was incorporated in the ladder logic program. Once a request is made for a job to be loaded onto the conveyer, the timer starts timing. At the designated set point the timer stops. During the collection of time for set points, there was no function of interfacing of the P/C.

The queueing and scheduling computational task was performed for the purpose of loading the most appropriate job onto the conveyer according to shortest processing time (S.P.T.), by each of the three parts loaders. Jobs were loaded onto the conveyer one at a time from each parts loader in sequence (i.e., one job from parts loader 1,

then one from 2 and one from 3), to avoid collision and damage on the conveyer. Hypothetically jobs enter the system and join different queues. Each queue has jobs that are made from the same blank. In addition, every job in every queue has its own set of operations that are needed to complete it. There are three physical parts loaders in the system, each of which can load one job at a time. To have a high throughput (minimize flowtime) in the system, the S.P.T. algorithm was implemented. From among all the jobs that are available for processing, the one which has the shortest processing time associated with it needs to be processed first. In terms of software, the number of queues can be increased to a very large number, with the smallest jobs in different queues competing for the parts loaders.

Set points for queueing and scheduling task

There were three set points for the test involving the queueing and scheduling computational task.

a) Set point 1: Time elapsed between request made for the parts loader to load a job onto the moving conveyer, and the cutting of the photobeam closest to the parts loader (see figure 5 page 29) by the block as it moved down the conveyer. This set point was found to be 2.41 seconds.

b) Set point 2: Time elapsed between request to load a job and cutting of the second photobeam just before the parts diverter (see figure 5 page 29) by the moving block. This set point was found to be 8.143 seconds.

c) Set point 3: Time elapsed between request to load a job and the block reaching the drilling station (i.e. after the indexing table comes to a stop after rotating 90 degrees). This set point was found to be 14.80 seconds.

The above readings represent an average of 14 readings each. The above set points are indicated by a horizontal line on a graph of response time against number of queues. With reference to these set points it is possible to visualize the time taken to load three blocks/jobs onto the conveyer, by the different approaches under consideration. It becomes practical to assess the throughput of the different approaches under different levels of computational tasks. For a particular size of a computational task, it might be possible via one approach to load the 3 jobs onto the conveyer before a particular set point is reached, and not by the other.

Set points for LP task

The basic understanding behind the algorithm for running of the LP task is as follows: after a job has been selected by the scheduling algorithm (by S.P.T.), it is loaded onto the conveyer for the purpose of processing. Associated with each part is a set of operational parameters which are either read at the the first photobeam or downloaded to the intelligent interfaced device, which then begins a process of optimization of those operational parameters by the revised simplex method. The results of such an optimization process need to reach the P/C within the time the job takes to reach either the second photobeam (where another process might be required to be performed on the part), or the drilling station- which are the two set points.

Two set points were selected for the Linear Programming task. These set points are indicated on a graph of response time against number of variables and constraints for the LP task.

a) Set point 1: Time elapsed between cutting of the first photobeam closest to the parts loader, and the cutting of the second photobeam before the diverter by the job, as it moved down the conveyer. This set point was found to be 5.69 seconds.

b) Set point 2: Time elapsed between cutting of the first photobeam by the job moving on the conveyer and the job reaching the drilling station. This set point was found to be 12.28 seconds.

P/C interface command calls

For the purpose of communication with the P/C over the network via the host software, several subroutines were used. There were essentially three classes of subroutines [16] that were used:

a) Session control subroutines: These subroutines need to be called by the application program to initiate and terminate the use of the TIWAY I library subroutines. The INIT subroutine reserves system resources used by the software package, and does port initialization.

b) Host adapter command code subroutines: The data necessary to create a host adapter command code command buffer, and data to be returned by the response to that command code are passed as arguments to these calls. The ACTVAT subroutine is used to logically connect a secondary to the network, before any communication is allowed with it.

c) NIM primitive subroutines: These subroutines correspond directly to a subset of the NIM primitives (universal command language instructions). They provide the applications programmer with a natural interface to the primitive. The arguments correspond directly to either information required to create the request buffer or data returned in response to the request. The TIREAD subroutine reads consecutive memory locations from a specified NIM-based attached device. The TIPUT subroutine is used to write to consecutive memory locations in a specified NIM-based attached device. The DEFBLK subroutine is used to define P/C memory locations. The

GATHER subroutine is used to gather data from previously defined data acquisition blocks.

In the BASIC module, CALL is the statement used to invoke subroutines that obtain information from, or send information to, the P/C. Of the five available subroutines, the following two were used:

a) The PCREAD subroutine is used to transfer data from the P/C to the BASIC module.

b) The PCWRITE subroutine is used to write to memory locations within the P/C.

4.3 Queueing & Scheduling test program development

The algorithm developed for performing this test is illustrated in Appendix G. The R.L.L. program (Appendix H) in the P/C is executed and it runs continuously (P/C in the run mode). The interfaced device reads (in its own running program) the appropriate memory location in the P/C to see if a request has been made for a job to be loaded onto the conveyer. The instant it reads the request, it runs through the queueing and scheduling algorithm (in a higher level language) to find the most suitable job to be loaded onto the conveyer, via the parts loader. When the processing is over the interfaced device writes to the designated output to turn on the first pneumatic parts loader. The write command places a value of '1' in that particular memory location. The task of turning off the parts loader is handled by the ladder logic program in the P/C. As soon as the first job is pushed out, the algorithm starts

again to load a job from the second parts loader, and subsequently the third parts loader.

The timing for the test starts as soon as there is a request for loading the first job onto the conveyer. This is performed by the P/C program. As soon as the third job is pushed out by the third parts conveyer, the P/C program turns the timer off. The inputs to the queueing and scheduling program in the interfaced device, are the number of queues and jobs per queue.

Subroutines developed for the Scheduling algorithm: The subroutine RAND returns with a random number. Selection of this subroutine was significant because the same subroutine needed to be used on two different machines (BASIC module & IBM-PC) in two different languages (FORTRAN & BASIC). If the random numbers generated had been different by the different hardware (ie. the inhouse random number generators of each individual machine), then the resulting scheduling problem, and especially the LP problems generated would have been different. If so, an improper evaluation of the different approaches would have occured, because different machines would have been running different problems. With a little modification in declaration for the different machines, this subroutine returned random numbers in the same order for every test performed in this thesis.

The subroutine QUE returns M queues with N jobs in each queue. The routine also calls subroutine PENAL which provides tardiness penalty costs. Finally it places the job with the shortest processing time in each queue as the first element of that queue.

Subroutine PENAL generates tardiness penalty costs in terms of processing time for every job in every queue.

4.4 LP test program development

The algorithm developed to perform this test is illustrated in Appendix G. A separate RLL program was loaded into the P/C and run. The interfaced device reads (in its own running program) the appropriate memory location in the P/C to see if a job has crossed the first photobeam. If it has, the interfaced device runs through the LP algorithm, to optimize the operational parameters associated with a particular job. The P/C program starts the timer as soon as the first photobeam is cut. At the end of the processing of the LP problem, the interfaced device writes to a specified memory location in the P/C to turn off the timer.

The inputs to the LP program in the interfaced device are the number of constraints, number of variables and a flag to indicate whether data needs to be manually entered, or generated internally. For the purpose of this thesis all the problems have been generated internally.

Subroutines used for the Linear Programming problem: Subroutine RINPUT returns with all the numerical values associated with the LP problem- the coefficient matrix, the cost matrix and the right hand side matrix.

Subroutine IBFS generates the initial basic feasible solution. Subroutine ENTER determines the entering variable by calculating the maximum Z(j)-C(j) value, and sets a flag when an optimum solution is found. Subroutine YK calculates the updated entering column and terminates the program if there is an unbounded solution to the problem generated at a particular iteration. Subroutine UPDATE performs the minimum ratio test and updates the basis inverse, RHS and duals. Subroutine SOLUTION prints the solution of the LP problem if an optimal solution has been reached. Subroutine RAND generates random numbers which are exactly similar to those generated by the same subroutine on the BASIC module and in the BASIC (compiled and interpreted) language, in the host computer.

4.5 Test program development for data transfer capability

In the industrial environment, the P/C runs some control program when there is need for interfacing it with some other processor, running parallel to it. During the running of the tests for data transfer capability, the ladder diagram shown in Appendix H was being run on the P/C. The same RLL program was run for all tests, because change in size of the program would cause a change in scan time of the P/C, and give erroneous results for response times.

Data transfer capability for consecutive word memory locations: As described earlier, some of the word data points in the P/C memory are V, WX, TCC, TCP etc. The BASIC module has a rated capability of reading 30 consecutive words per scan of the P/C. Therefore, the stepwise readings of the number of consecutive word data points

accessed were taken in multiples of 30 up to 960. The tests were run in FORTRAN in the host PC on the network, and in allowable interpreted BASIC on the BASIC module.

With the CALL PCREAD statements on the BASIC module, it is possible to read word or discrete data points without changing any declarations. This is possible because in the BASIC module, all variables and constants are represented in 48 bits. The only change that needs to be made is in the CALL statement.

Only one PCREAD statement can be called per scan of the P/C, and if more data than can be transferred in a single scan (for example, more than 30 words per call) is requested, the remaining data is transferred in subsequent scans. The program in the BASIC module continues only after all the information has been transferred by the PCREAD statement. The actual test program consisted primarily of three CALL statements for the BASIC module (Appendix I). PCWRITE for turning the external timer 'on', followed by PCREAD and immediately followed by PCWRITE again for turning the timer off.

Over the network the subroutine called was TIREAD. With this statement it is possible to access a maximum of 134 consecutive words at 28 words per scan of the P/C. To access more than 134 consecutive words, another TIREAD subroutine needs to be called for up to another 134 words. For this test a maximum of 960 consecutive words needed to be used. This was possible by using 8 consecutive TIREAD statements (Appendix J). In the actual test by this approach, the number of CALL statements depended on the number of word data points that needed to be accessed. Two TIPUT calls were used, one before the first TIREAD and one after the last TIREAD, to start and stop the external timer respectively.

Data transfer capability for consecutive discrete data points:

Discrete data points in the P/C memory are X, Y and C. The BASIC module has a rated capability of reading 26 consecutive discrete data points per scan of the P/C. Therefore, for the purpose of this test, the number of data points accessed were taken in multiples of 26 up to 896. As in the tests for consecutive word data points, FORTRAN was used to program the host PC, and interpreted BASIC to program the BASIC module.

The test for consecutive discrete data points on the BASIC module was administered in exactly the same way as it was for word data points. The only change was in the PCREAD statement requesting discrete data points.

As noted earlier, each TIREAD subroutine call over the network can access up to 134 word memory locations. Since there is a facility in the network software to read discrete data points in a compacted format, (i.e. read one discrete data point for each bit of information read) it was adopted for this test. Reading 28 words (one scan of the P/C) reads 448 discrete data points, and reading 56 words (two scans of the P/C) reads 896 (the top limit for this test) discrete data points. Therefore, the actual test by the network approach used primarily three subroutine calls. Two TIPUT statements to start and stop the external timer, and one TIREAD call accessing the required number of discrete data points in compacted form.

Data transfer capability for non-consecutive word and discrete data points:

As stated earlier, in most industrial applications it is quite possible to have a need for accessing different types of memory locations in the P/C, both word and discrete data points that are non-consecutive. Over the network it is possible by two ways, a) a

succession of TIREAD calls, one for each data point needed to be accessed, or b) use of the GATHER subroutine call. In the BASIC module it is possible only by repeating the PCREAD call (as in TIREAD), once for each different data point needed to be accessed.

The GATHER subroutine can access data blocks which have earlier been defined. The DEFBLK subroutine is used to define P/C memory locations. The define primitive that is accessed by the DEFBLK subroutine can define up to 32 random blocks of data element types. Therefore the GATHER subroutine can access a total of 32 different word or discrete data points via one subroutine call, and the test was set up to access from 1 to 32 non-consecutive memory locations in steps, via GATHER and TIREAD subroutine calls over the network, and with the PCREAD subroutine call on the BASIC module. For setting up the GATHER subroutine call, it is also required to call a BLDMSK subroutine which builds a mask from a list of block numbers.

As in the tests for discrete and word data points over the network, two TIPUT calls were needed, one before the GATHER call and one immediately after, to start and stop the timer respectively (Appendix K). For using TIREAD for the above test, as many TIREAD calls were placed between the two TIPUT calls as locations were needed to be accessed (word or discrete data points). For 32 different locations to read, 32 TIREAD calls were placed between the two TIPUT calls.

For this test on the BASIC module, two PCWRITE calls were used to start and stop the timer. The required number of PCREAD calls (one for each data point) were placed between the two PCWRITE calls (Appendix I).

5.0 Chapter 5. Analysis of results

5.1 Introduction

In this chapter the results of the tests performed are discussed. The memory limitations of the BASIC module are interpreted, and the assumptions and limitations of the tests are also presented. An analysis of the two approaches from the point of view of cost is outlined. Additional features of the host computer and the network are also detailed. All results are shown graphically along with the text, and also tabulated in Appendix L.

Assumptions: It is assumed that similar results (comparison), would emerge for the commands by both approaches for writing to memory locations in the P/C. It is also assumed that the computational tasks performed for this work, are representative of actual industrial applications.

It is known from the specifications of the hardware used that the clock rate on the processor of the BASIC module is 12 Mhtz, and that on the processor of the IBM-PC is 4.7 Mhtz (i.e., the processor on the BASIC module is faster than that on the IBM-PC).

5.2 *Data Transfer Capability*

The results of the tests for data transfer capabilities between the P/C and the BASIC module and host computer on the TIWAY I network are presented in the order they were obtained.

5.2.1 Consecutive word data points

The comparison of the two approaches, for accessing consecutive word memory locations, reveals that the BASIC module is much faster than the host computer over the network (Figure 6 on page 52). Over the entire range of values used for this test, the host computer took over four times as much time as the BASIC module.

The primary reason for this is the processing power of the different interfaced devices, to process each interface call. The baud rate between the host computer and the P/C also adds some overhead to total processing time. As stated earlier, the baud rate between the host and the host adapter is 9.6K, and that between the host adapter and the NIM (network interface module) is 115.2K. Between the NIM and the

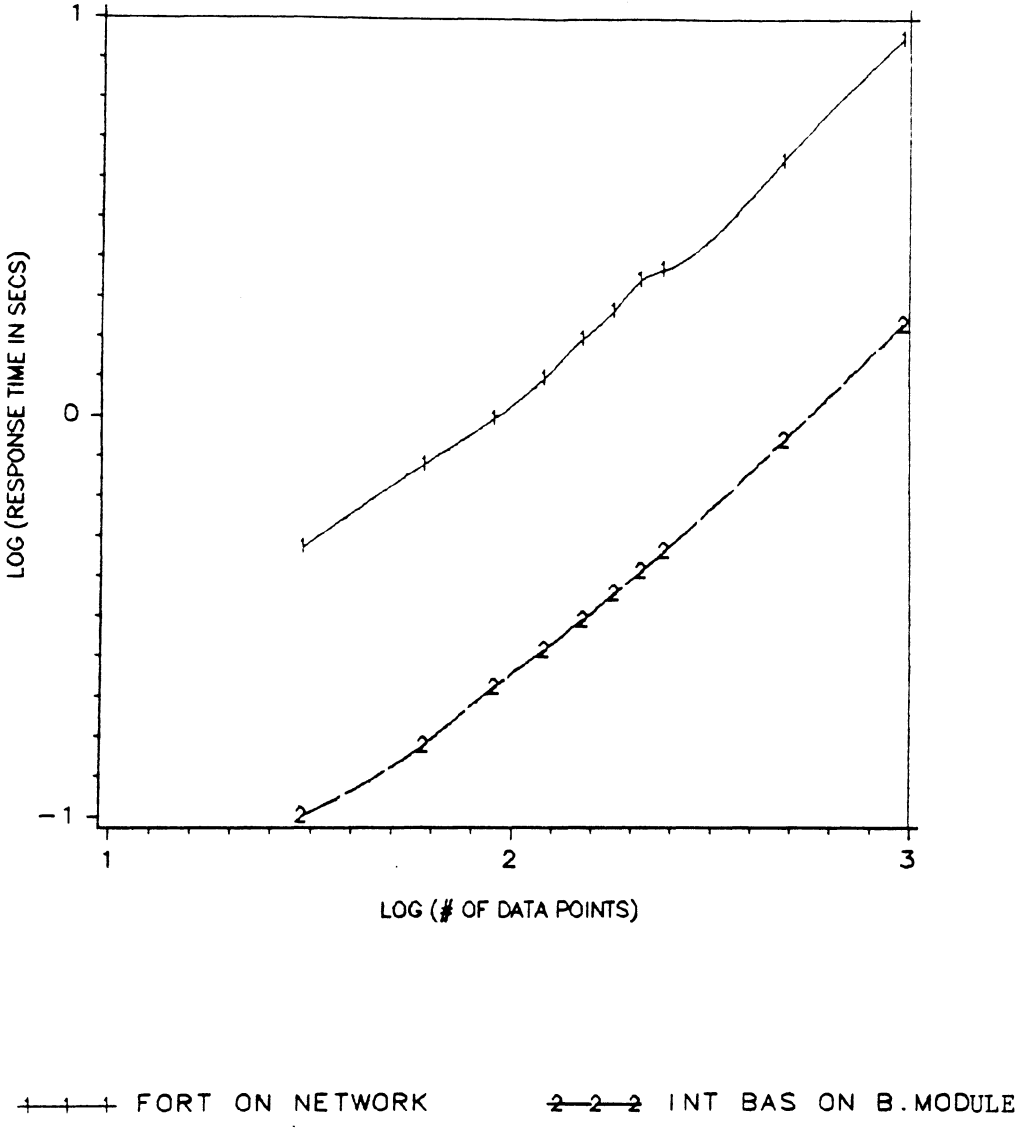


Figure 6. Response time for consecutive word data points.

P/C, the communication takes place at bus speeds (on the I/O base of the P/C). With the BASIC module all communication takes place at bus speeds, because it sits on the bus. The second reason is that several TIREAD calls were used to be able to read up to 960 word locations with the host computer, as opposed to one PCREAD call by the BASIC module. The more the number of TIREAD calls, more the number of times information is transmitted at 9600 bps back and forth.

5.2.2 Consecutive discrete data points

The tests for comparison of capability for reading consecutive discrete data points revealed that, for up to 126 data points, the BASIC module was faster, but beyond that the host on the network performed better (Figure 7 on page 54). At 896 data points (the largest value tested), the BASIC module took almost three times as much time as the host computer.

As stated in the previous chapter, the compacted format was used (1 bit for every data point) to read the consecutive discrete data points over the network. Since each TIREAD call can retrieve 28 words (448 discrete data points) per scan of the P/C, it takes just one scan to read 448 discrete data points and two scans to read 896 data points. On the other hand, it takes the BASIC module over 17 scans of the P/C to read 448 discrete data points (at 26 per scan of the P/C), and over 34 scans to read 896 data points. Initially the BASIC module is faster because it has a faster processor, and few scans of the P/C are involved. In addition, information travels at bus speeds, as opposed to a baud rate of 9.6K by the TIREAD call. However, as stated above with

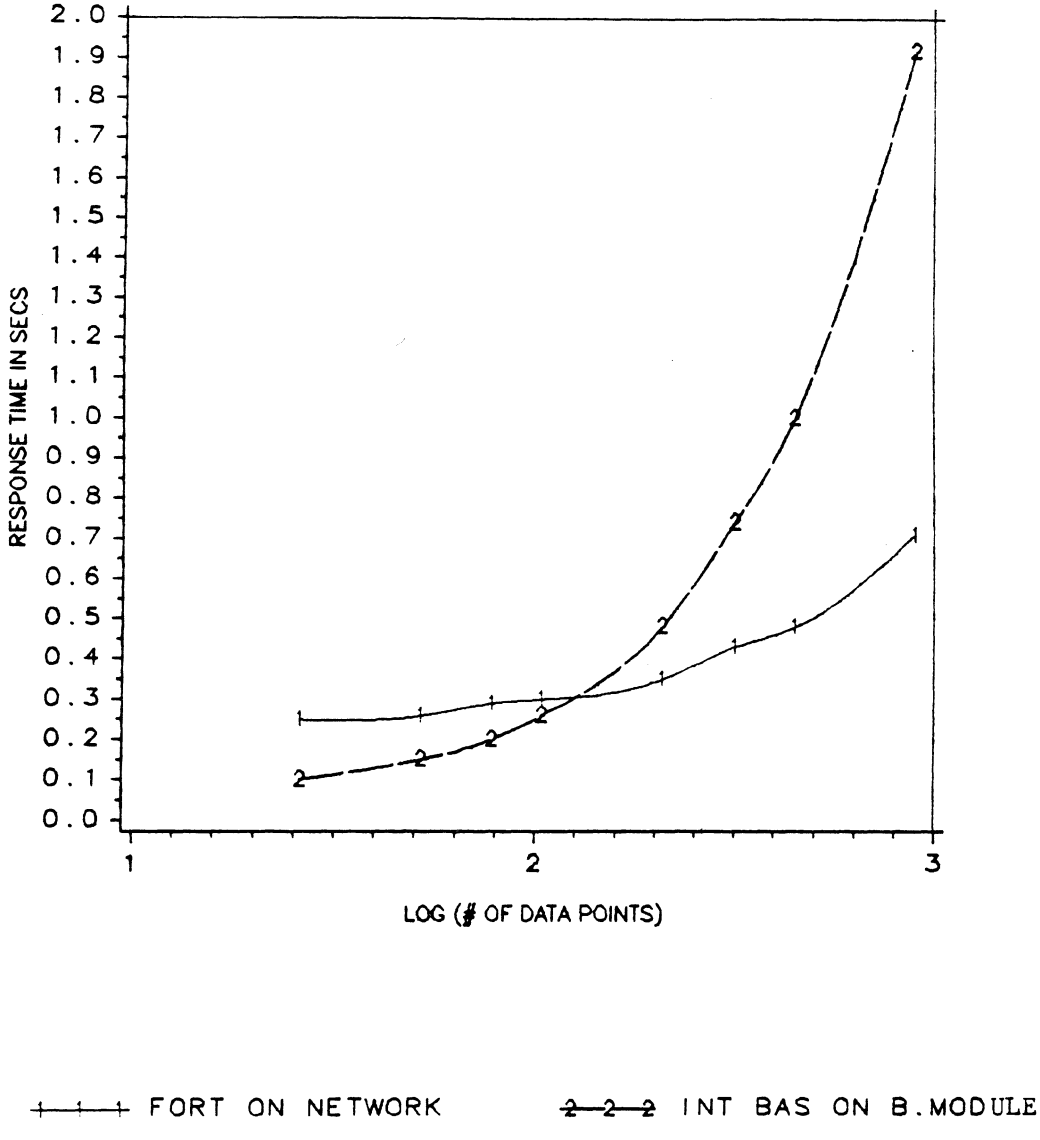


Figure 7. Response time for consecutive discrete data points.

increasing number of data points accessed (increasing the number of P/C scans for the BASIC module significantly), the host computer becomes more efficient.

5.2.3 Non-consecutive data points

The results of comparison for data transfer capability for non-consecutive discrete and word data points are displayed in Figure 8 on page 56. Earlier it had seemed that the possibility of being able to access 32 different non-consecutive data points by one GATHER subroutine call would be a very significant advantage for the network. However, the results as shown indicate that an equivalent number of PCREAD calls on the BASIC module (i.e. 10 PCREADs for accessing 10 different non consecutive data points for example) is more than twice as fast as using the GATHER call on the network, and almost three times as fast as using TIREAD calls (i.e. one TIREAD call for each data point accessed) over the entire range of tests (from 1 to 32).

Thus, the data transfer capability of the BASIC module is superior to that of the host computer on the network, except when it comes to accessing over 126 consecutive discrete data points.

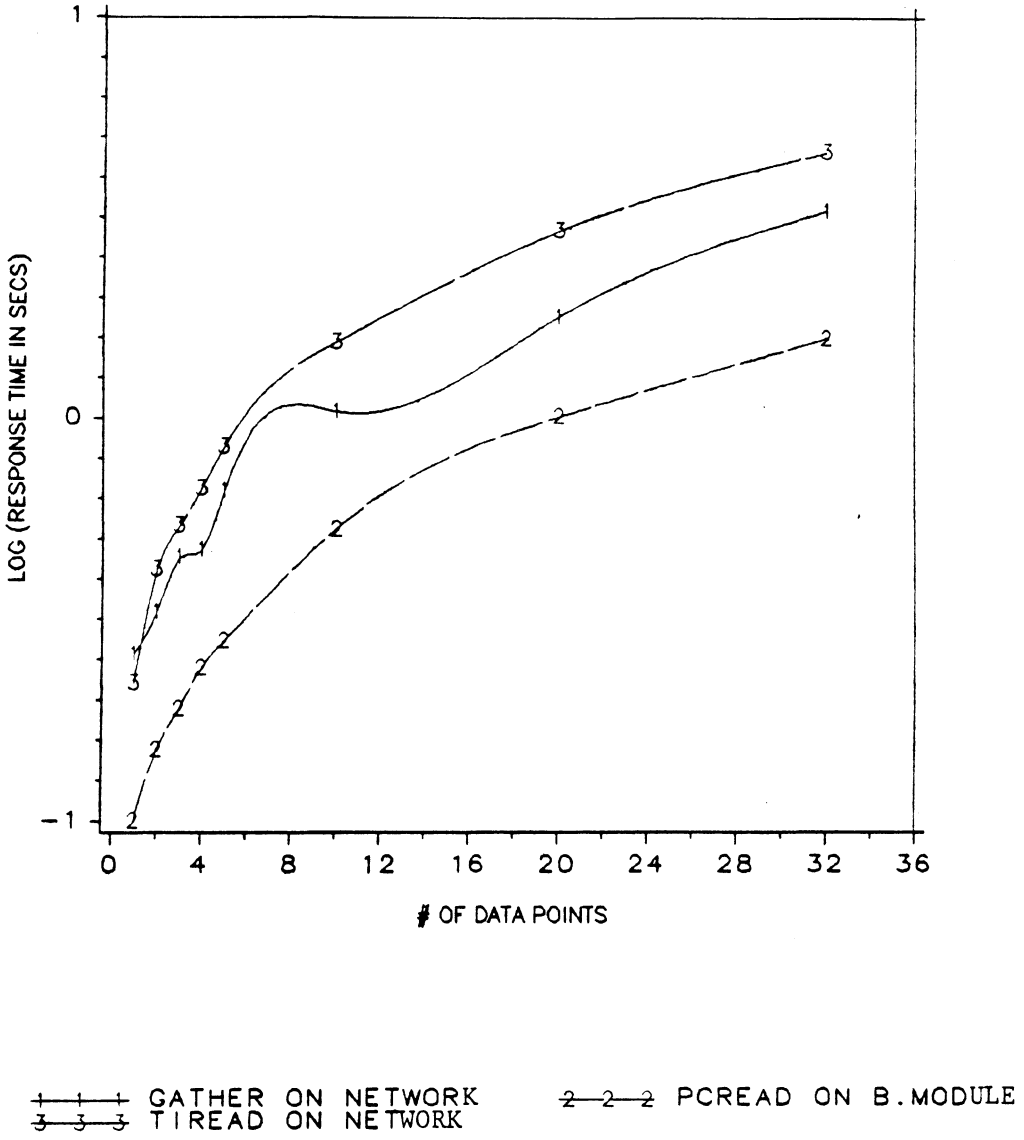


Figure 8. Response time for non-consecutive data points.

5.3 *Computationally Intensive Tasks*

5.3.1 Linear Programming Tasks

The results of this computationally intensive task are illustrated in Figure 9 on page 58 (where time readings indicate time interval for the interfaced device to complete a specific task and send the response back to the P/C), show that processing on the BASIC module takes more than 11 times as much time for processing the same task as running FORTRAN in the host PC over the network, and over 1.6 times as much as compiled QUICKBASIC. However, running interpreted BASIC in the host PC on the network takes more than 7 times as much time for processing, as does the BASIC module.

5.3.2 Queueing and Scheduling tasks

As detailed in the previous chapter, the algorithm for this task runs three times for the purpose of loading one job from each parts loader. The time values obtained for the task constitute a cumulative time for performing three runs of the same algorithm each time. The results of the computational task of this nature are shown in Figure 10 on page 59. For a problem size of 5 queues (with 10 jobs per queue), the

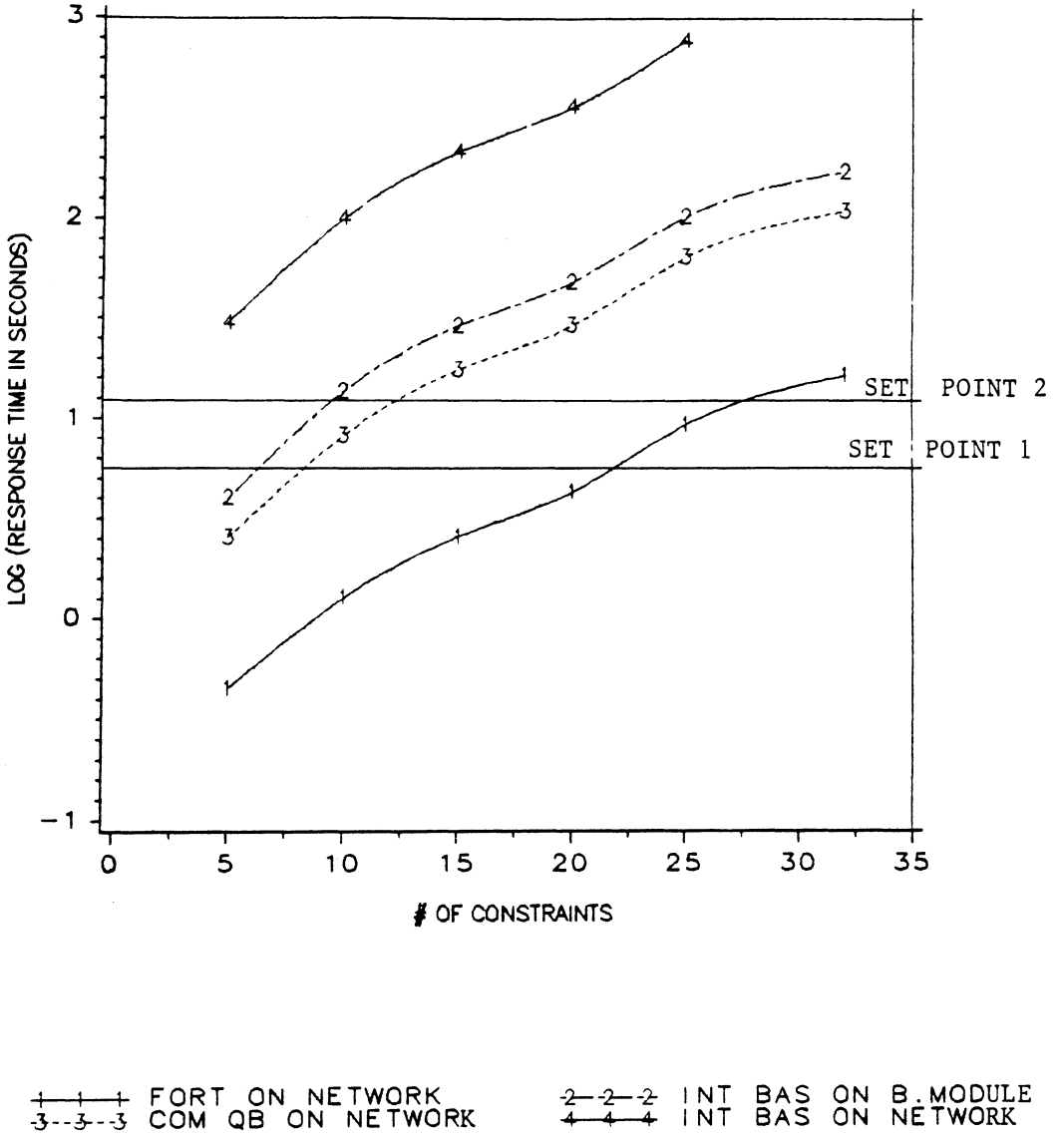


Figure 9. Response time for LP task.

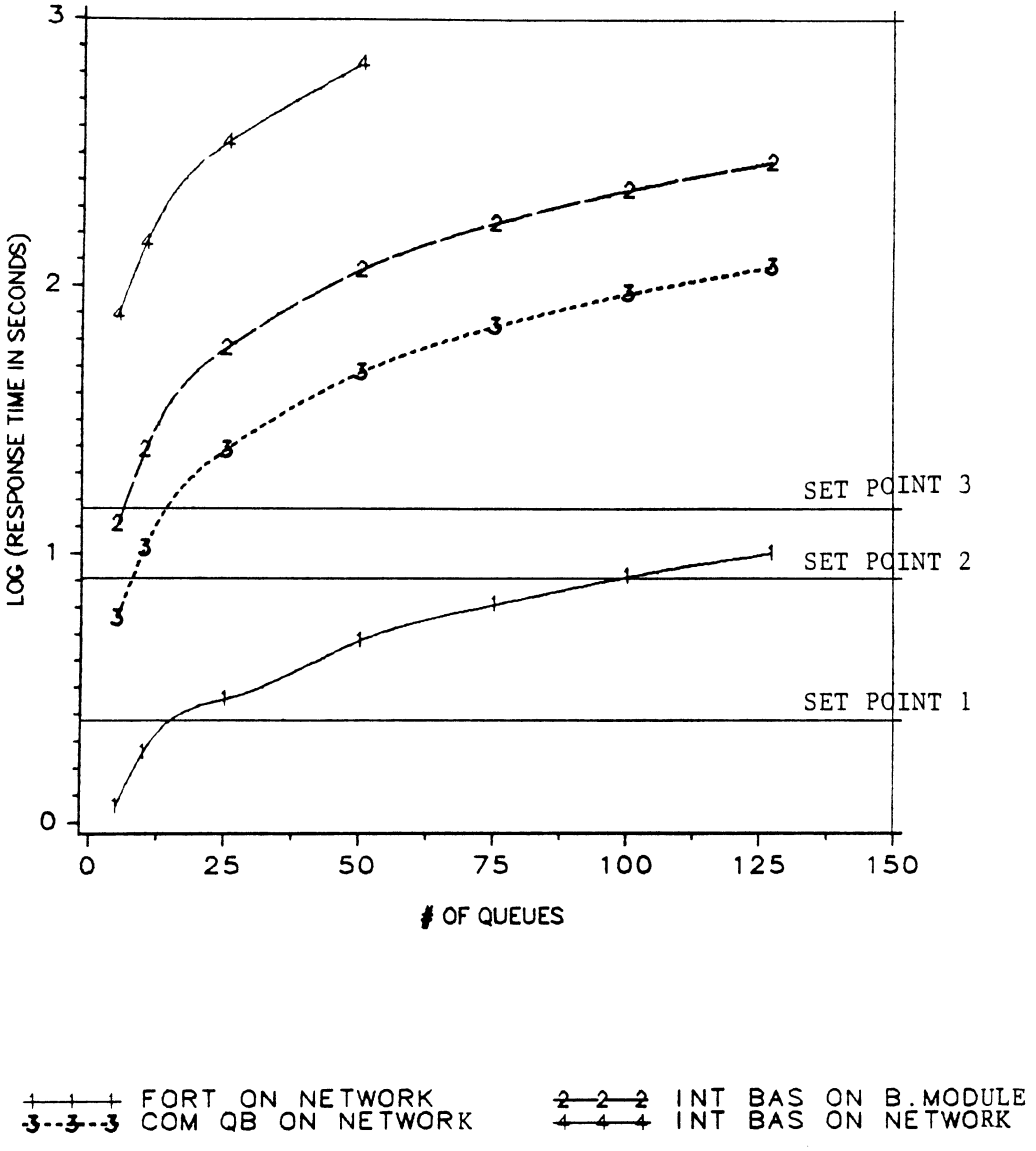


Figure 10. Response time for queueing and scheduling task.

BASIC module takes over 11 times as much time as does running FORTRAN in the host PC over the network. This order of magnitude increases as the problem is increased in size. With 25 queues (10 jobs per queue) it is over 20 times, and at the largest computation for this test, 127 queues (10 jobs per queue), the magnitude is almost 29 times. The above phenomena probably occured because the same algorithm is run three times for each test run for a total time count. With an increasing size of the problem, the time taken by the BASIC module is more than that by FORTRAN on the network for each iteration. For three iterations this gap in processing capability widened even more, as seen in the results.

However, a similar phenomenon is not observed in the comparison of compiled QUICKBASIC in the host PC over the network and the BASIC module. Interpreted BASIC on the BASIC module takes 2.2 times as much time as does QUICKBASIC on the network for a problem size of 5 queues (10 jobs per queue), and increases very gradually up to 2.4 times for a problem size of 127 queues (10 jobs per queue). Therefore, it can be concluded that the results obtained by using FORTRAN for this test were not totally dependent on the way the test was set up (algorithm dependent). It could also be due to the language that was being used (FORTRAN). However, there was no marked increase in the magnitude for processing time during the tests for the LP task in FORTRAN. For the LP task, the BASIC module takes between nine and 11.2 times as much time as does FORTRAN on the network over the entire range of tests conducted. Therefore, it is a combination of the algorithm and how the processors process the language that have given the above results.

For both the tests for computationally intensive tasks, the BASIC module is substantially slower than the host computer (for compiled languages), even though the

processor on the BASIC module is over two times faster. It is evident that the cause is the running of allowable interpreted BASIC on the BASIC module. Interpreted BASIC over the network takes about six times as much time as interpreted BASIC on the BASIC module. This is the result of the processing power of the respective processors and the baud rate for information flow between the interfaced device and the P/C.

5.4 Memory Considerations

The memory on the BASIC module is 28K and is not expandable. If additional memory is needed, users typically buy another similar BASIC module. The tests for computationally intensive tasks were carried on till the problems became too large for the memory on the BASIC module. The largest size of a Linear Programming problem that could be solved was one of 40 variables and 32 constraints. On the same token the largest queueing and scheduling problem that could be solved was one of 127 queues with ten jobs per queue. For problems of larger dimensions than those just stated, a memory overflow occured. A total of up to 128 unique variable names is allowed in the BASIC module. The memory on the PC is very large compared to that available in the BASIC module.

5.5 Cost Considerations

The BASIC module costs approximately $1200. There is need for a non-intelligent terminal to program it. A VPU200 can be used once it has been initialized with the BASIC operating system disk. An IBM-PC can also be used once it is configured as a dumb terminal. The programming device may be disconnected once the program has been loaded and run, and used for other functions. It would be advisable to use a PC to program the BASIC module, because the cost of a VPU200 with the BASIC operating system can run over $6000. It is possible to program a P/C with a PC with use of software such as the TISOFT2, which runs for approximately $250. This cost would be common for both the approaches.

There are several costs involved for the network. The Network Interface Module (NIM) costs approximately $950 and the host adapter costs approximately $1350. The TIWAY I host software runs for about $1150, and there has to be a dedicated host computer. Apart from this there is the cost of cabling between the host and the P/Cs on the network. There is also more investment in time and talent for learning and using the network.

However, the network has vast capability and power. One host computer can control up to 254 secondaries on the network. These secondaries could be from any series 500, 5TI,102,103 and the PM550. The BASIC module is only useful with the 520/530 P/Cs. The host computer, especially if it supports a multitasking environment can be used to format reports and give graphical displays, as it communicates with the P/Cs. With the host computer it is possible to turn off a particular secondary (P/C) on the

network. It is not possible to do the same by the BASIC module. The host computer can be used as a cell controller, and pass shop floor information upwards to the plant computer.

6.0 Chapter 6. Conclusions and Recommendations

This work was performed specifically on the Texas Instruments hardware and software described. The values for response time that are tabulated are specific for the setup used for this research. They can be different for different lengths of cable from PC to host adapter and from P/C to the control system. The P/C program size also has a substantial effect on the actual response time. In this work however, a comparison has been performed between the two approaches of P/C interfacing with respect to certain attributes, using the same setup.

This work can help a potential user to evaluate his particular application function, with respect to need for transferring and accessing information from a P/C, and running computational tasks in parallel to the P/C. It has also defined limitations in both the systems, specifically memory in the BASIC module, for two typical industrial applications oriented tasks, a) linear programming and b) queueing and scheduling.

Based on the results of this work, the BASIC module need only be used where up to medium size computation is required to be performed in parallel to the P/C, with the

intended application needing low to moderate response times for computationally intensive tasks. This is owing to its inherent memory limitation of 28K bytes, and the use of interpreted BASIC. The capability of the its processor and the fact that it sits on the I/O bus, allows it to process P/C interface subroutine calls much faster than that by a host on the TIWAY I network. The BASIC module can also be of substantial use if the resources of the host computer on the network are not available for computation, or for operation with specialized devices such as bar code readers or CRTs. This can occur if the host is communicating with other P/Cs, higher level computers, or being used for other purposes.

As was evident, the host computer on the TIWAY I network is the superior alternative in terms of ability to store and process large and small computational tasks, owing to greater memory storage capabilities and the use of compiled languages.

The host and the network cost about four times as much as the BASIC module. However, for a large facility with several P/Cs, or looking to future expansion and use of P/Cs, the network can serve tremendous benefits, especially for plantwide integration and for reports and displays.

Future work could pursue similar tests using other higher level languages in host computer over the network, eg. 'C' or 'Pascal'. There would probably be an increase in processing speed for computational tasks by an order of magnitude over FORTRAN. It would also be interesting to note the performance for data transfer and computational capability, by using a computer (eg. an IBM-PC/AT) as a host to the network, which allows communication through the serial port at a baud rate of 19.2k, and which has a faster processor and more memory. It would also be interesting to evaluate the SRTC (send and receive task codes) subroutine in the BASIC module.

This subroutine can change the P/C program because it sends operational codes (rather than values) to the P/C.

A similar series of tests could be done on similar products available from manufacturers other than Texas Instruments, for example Allen Bradley, General Electric, Honeywell etc. The same tests as used for this work could be used for them too.

7.0 References

1. Asfahl, R. C., Robots and Manufacturing Automation , John Wiley and Sons, New York, 1985.
2. Carr, H. A., "Distributed Control System Features Built-In P/C Capability", Programmable Controls , May/June 1987, pp 97-100.
3. Galgoci, C. B., "Integrating a Flexible Manufacturing System with Programmable Controller Ladder Logic through Simulation", M.S. Thesis , Virginia Polytechnic Institute and State University, 1986
4. Gibbons, W. M., "Selection and Design Issues for Industrial Communications", Proceedings-15th Annual International Programmable Controllers Conference and Exposition , April 1986, pp 1-6.
5. Gosset, S. B., "Intelligent Communications Handler for Flexible Manufacturing Lines", M.S. Thesis , Virginia Polytechnic Institute & State University, 1986.
6. Gould, L., "The Ubiquitous Versatile Programmable Controller", Managing Automation , November 1986, pp 41-47.

7. Hagen, R. S., "PC to Personal Computer Interfaces", Proceedings-14th Annual International Programmable Controllers Conference and Equipment Display , April 1985, pp 61-68.

8. Hill, G. D. and Deane, R. H., "Integrating the Programmable Controller and the Personal Computer: A Manufacturing Application", Proceedings-15th Annual International Programmable Controllers Conference and Exposition , April 1986, pp 369-377.

9. Host Software Programming Manual , Texas Instruments Manual No. TIWAY 8105.

10. Jasany, L. C., "Tying the Factory Together with PCs and Networks", Production Engineering ,April 1984, pp 52-62.

11. Jones, C. T. and Bryan, L. A., Programmable Controllers: Concepts and Applications , International Programmable Controls, Inc., Atlanta, 1983.

12. Jurgen, R. K., "Industry's Workhorse Gets Smarter", IEEE spectrum , February 1982, pp 34-38.

13. Kinsey, G. H., "Integration of Computers into the Programmable Controller Environment: A Management Overview", Proceedings-15th Annual International Programmable Controllers Conference and Exposition , April 1986, pp 229-236.

14. McLoughlin, J. M., "A Systematic Approach to Programmable Controller Controlled Machine Monitoring Using Microcomputers", Proceedings-15th Annual International Programmable Controllers Conference and Exposition , April 1986, pp 295-306.

15. Meinhold, T. F., "Programmable Controllers: An Overview of Sizes and Capabilities plus Guidelines on Selection, Installation, Maintenance, and Use", Plant Engineering , November 23 1983, pp 52-67.

16. Model 530 Programmable Controller , Texas Instruments Manual No. 530-8101, February 1983.

17. Model 510 Programmable Controller , Texas Instruments Manual No. 530-8101, February 1983.

18. Pinto, J. A., "PC-to-P/C Communications Through Multi-Processors", Programmable Controls , Jan/Feb 1987, pp 95-98.

19. Press, W. H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W. T., Numerical Recipes, the Art of Scientific Computing , Cambridge University Press, New York, 1986

20. Programmable BASIC Module User's Guide , Texas Instruments Manual No. 500-8107, July 1986.

21. Roeder, W.H., "Communications Between Programmable Controllers in the Industrial Environment", IEEE Transactions on Industry Applications , Vol. IA-20, No. 3, May/June, 1984, pp 504-509.

22. Snader J. A., "Toward the Development of a Universal Programming/Documentation System for Programmable Controllers on a Host Microcomputer", M. S. Thesis , Virginia Polytechnic Institute and State University, 1985

23. TIWAY I Host Adapter User's Manual , Texas Instruments Manual No. TIWAY 8102.

24. TIWAY I Network Interface Module , Texas Instruments Manual No. 500-8103.

25. TIWAY Systems Manual , Texas Instruments Manual No. TIWAY-8101.

Appendix A.

THIS PROGRAM GENERATES AND SOLVES LINEAR PROGRAMMING PROBLEMS IN FORTRAN

```
C*********************************************************************
C*********************************************************************
C
C     THIS PROGRAM GENERATES AND SOLVES LINEAR PPRGRAMMING PROBLEMS
C     IN THE FORTRAN LANGUAGE BY
C     USING THE REVISED SIMPLEX METHOD
C
C     THE LP IS OF THE FORM : MIN CX, SUBJECT TO AX < = B , X > = 0
C                                    ( B NONNEGATIVE )
C     M = # OF ROWS OF A (# OF CONSTRAINTS)
C     N = # OF COLUMNS OF A (# OF VARIABLES)
C     COST(N), B(M), A(M,N) ARE ORIGINAL DATA
C     BASIS(M,M) IS  BASIS INVERSE ; RHS(M) IS UPDATED RHS-VECTOR
C     DUAL(M) IS VECTOR OF DUAL VARIABLES ; Y(M) IS UPDATED ENTERING COL
C     JBASIC(M+N) IS  +I IF VARIBLE J IS BASIC AT THE I TH POSITION,
C                 -J OTHERWISE
C
C*********************************************************************
C*********************************************************************

      IMPLICIT REAL   (A-H,O-Z)
      DIMENSION COST(90),B(45),A(45,90),BASIS(45,45),RHS(45),
     *          DUAL(45),Y(45),JBASIC(120),X(90)
C
      CHARACTER RI135(135)
C      INTEGER*2 RI135(135)
      INTEGER       WI4(4),WI135(135)
      INTEGER*2 RI4(4),WILEN,RILEN,NN,ISTAT,SSTAT,HWY
      INTEGER*2 XTN,RSPLEN,OUT(50)
      INTEGER*2 TAG(5),TA1(5)
      ISTOP=0
      ITER=1
      ITERM=200
      IX=117
      RSPLEN=135

C      TAG='#0101040049'
      TAG(1)=0
      TAG(2)=1
      TAG(3)=1
      TAG(4)=03
      TAG(5)=36
```

```
      TA1(2)=1
      TA1(3)=1
      TA1(4)=04
      TA1(5)=135
      WILEN=1
      WI4(1)=1
      SSTAT=0
      HWY=1
      XTN=0
      ISTAT=0
      NN=1
      MM=1
      WI135(1)=1
C
C
C     READS THE NUMBER OF VARIABLES 'N' AND CONSTRAINTS 'M' DESIRED
C     IDATA = 1 IF YOU WANT TO ENTER DATA MANUALLY
C
C
      READ(*,*) M,N,IDATA
      CALL INIT(ISTAT,XTN,'P1,9600,7,1,E,3,A')
C
C
C     THIS IS A SESSION CONTROL SUBROUTINE. IT MUST BE CALLED BY
C     ANY APPLICATION PROGRAM TO INITIATE AND TERMINATE THE
C     USE OF THE 'TIWAY' LIBRARY SUBROUTINES
C
C
      CALL ACTVAT(ISTAT,XTN,HWY,WILEN,WI4,RILEN,RI4)
C
C     THIS IS A HOST ADAPTER COMMAND CODE SUBROUTINE
C     BEFORE ANY COMMUNICATION IS ALLOWED WITH A SECONDARY
C     IT NEEDS TO BE LOGICALLY CONNECTED TO THE NETWORK
C     THIS IS DONE USING THE ACTVAT SUBROUTINE
C
C
 19   CALL TIGET(ISTAT,XTN,TAG,NN,RSPLEN,RI135,SSTAT)
C
C
C     TIGET IS A NIM PRIMITIVE SUBROUTINE. IT IS USED TO
C     READ CONSECUTIVE MEMORY LOCATIONS FROM A SPECIFIED
C     NIM-BASED ATTACHED DEVICE
C
C
      DO 20 K=1,RSPLEN
      OUT(K)= ICHAR(RI135(K))
 20   CONTINUE
      IF (OUT(1).EQ.1) THEN
C     READ IN THE INPUT DATA
        CALL RINPUT(COST,B,A,M,N)
       ELSE
        GOTO 19
      ENDIF
C
C     GET THE INITIAL BASIC FEASIBLE SOLUTION
      CALL IBFS(BASIS,DUAL,B,JBASIC,RHS,OBJ,M,N)
C
 500  CONTINUE
C
C     CALCULATE THE MAX [Z(J)-C(J)] VALUE (DETERMINE THE ENTERING VAR)
      CALL ENTER(DUAL,A,COST,JBASIC,IDUM,JENTER,M,N,ISTOP,ITER)
      IF(ISTOP.NE.1) GO TO 1000
      ISTAT=-1
```

```
      CALL TIPUT(ISTAT,XTN,TA1,MM,WI135,SSTAT)
C
C     THIS SUBROUTINE IS USED TO WRITE TO CONSECUTIVE MEMORY
C     LOCATIONS IN A SPECIFIED NIM-BASED ATTACHED DEVICE
C
      CALL SOLN(JBASIC,RHS,B,DUAL,OBJ,M,N)
C     THIS SUBROUTINE PRINTS THE SOLUTION OF THE LP PROBLEM
      ISTAT=-1
      WI135(1)=0
      CALL TIPUT(ISTAT,XTN,TA1,MM,WI135,SSTAT)
      GO TO 9000
 1000 CONTINUE
C
C     CALCULATE THE UPDATED COLUMN OF THE ENTERING VARIABLE
      CALL YK(Y,BASIS,A,JENTER,M,N,ISTOP,ITER)
      IF(ISTOP.EQ.1) THEN
        GO TO 8900
       ELSE
        GOTO 8800
      ENDIF
 8900 ISTAT=-1
      WI135(1)=1
      CALL TIPUT(ISTAT,XTN,TA1,MM,WI135,SSTAT)
      WRITE(*,7000) ITER
 7000 FORMAT(//,5X,'UNBOUNDED SOLUTION. ITER=',I4)
      WI135(1)=0
      CALL TIPUT(ISTAT,XTN,TA1,MM,WI135,SSTAT)
      GOTO 9000
C
C     DO THE MIN RATIO TEST, UPDATE THE BASIS INVERSE, RHS & DUALS
C
 8800 CALL UPDATE(BASIS,DUAL,Y,RHS,JBASIC,IDUM,OBJ,JENTER,M,N,ITER)
      IF(ITER.LT.ITERM) GO TO 8500
      WRITE(*,8000) ITER
 8000 FORMAT(/,5X,'TERMINATION DUE TO THE LIMIT ON ITERS. ITER=',I4)
      CALL SOLN(JBASIC,RHS,B,DUAL,OBJ,M,N)
      GO TO 9000
 8500 CONTINUE
      ITER=ITER+1
      GO TO 500
 9000 CONTINUE
      STOP
      END
C
C*********************************************************************
C     THIS SUBROUTINE READS THE INPUT EITHER BY GENERATIN
C     THE VALUES OF THE CONSTANTS OR BY MANUALLY ENTERING THEM
C*********************************************************************
C
      SUBROUTINE RINPUT(COST,B,A,M,N)
      IMPLICIT REAL  (A-H,O-Z)
      DIMENSION COST(90),B(45),A(45,90)
C
      IDATA=1 IF DATA IS AVAILABLE
      IX=117
      IF(IDATA.EQ.0) GO TO 2000
      READ(*,*) M,N
      READ(*,*) (COST(I),I=1,N)
      READ(*,*) (B(I),I=1,M)
      DO 1000 I=1,M
 1000 READ(*,*) (A(I,J),J=1,N)
```

```
      GO TO 4000
C
C     GENERATE THE PROBLEM
 2000 CONTINUE
      AVE1=20.
      AVE2=50.
      AVE3=100.
      RHO=0.6
      DO 3000 J=1,N
      CALL RAND(RAN1)
      COST(J)=AVE1*RAN1
 3000 IF(RAN1.GT.RHO) COST(J)=-COST(J)
      IF(COST(3).GT.0) COST(3)=-COST(3)
      DO 3020 I=1,M
C
      CALL RAND(RAN1)
      TEMP=RAN1
      B(I)=0.
      IF(TEMP.GE.0.1) B(I)=1.
 3020 CONTINUE
      DO 3030 I=1,M
      DO 3030 J=1,N
      CALL RAND(RAN1)
      A(I,J)=AVE1*RAN1
 3030 IF(RAN1.GT.RHO) A(I,J)=-A(I,J)
C
 4000 CONTINUE
C
      NLP=N+M
      N1=N+1
      DO 5000 I=N1,NLP
 5000 COST(I)=0.
      DO 6000 I=1,M
      DO 6000 J=N1,NLP
      A(I,J)=0.
      JJ=J-N
      IF(JJ.EQ.I) A(I,J)=1.
 6000 CONTINUE
C
 9000 FORMAT(5E14.4)
 9010 FORMAT(2I6)
      RETURN
      END
C
C*********************************************************************
C     THIS SUBROUTINE GENERATES THE INITIAL BASIC FEASIBLE
C     SOLUTION
C*********************************************************************
C
      SUBROUTINE IBFS(BASIS,DUAL,B,JBASIC,RHS,OBJ,M,N)
      IMPLICIT REAL  (A-H,O-Z)
      DIMENSION BASIS(45,45),DUAL(45),B(45),JBASIC(120),RHS(45)
C
      DO 1000 J=1,M
      DO 1000 I=1,M
      BASIS(I,J)=0.
 1000 IF(I.EQ.J) BASIS(I,J)=1.
      DO  2000 I=1,M
      RHS(I)=B(I)
 2000 DUAL(I)=0.
      N1=N+1
      NM=N+M
      DO 3000 I=1,N
```

```
3000 JBASIC(I)=-I
     DO 4000 I=N1,NM
     J=I-N
4000 JBASIC(I)=J
     OBJ=0.
     RETURN
     END
C
C*********************************************************************
C    THIS SUBROUTINE DETERMINES THE ENTERING VAR
C    BY CALCULATING THE MAX [Z(J)-C(J)]
C*********************************************************************
C
     SUBROUTINE ENTER(DUAL,A,COST,JBASIC,IDUM,JENTER,M,N,ISTOP,ITER)
     IMPLICIT REAL  (A-H,O-Z)
     DIMENSION DUAL(45),A(45,90),COST(90),JBASIC(120)
     IDUM=-10000000.
C
C    FIRST CALCULATE THE MAX REDUCED COST FOR REAL VARS (NON-SLACKS)
     DO 2000 J=1,N
     JJ=JBASIC(J)
     IF(JJ.GT.0) GO TO 2000
     Z=0.
     DO 1000 I=1,M
1000 Z=Z+DUAL(I)*A(I,J)
     TEMP=Z-COST(J)
     IF(TEMP.LE.IDUM) GO TO 2000
     IDUM=TEMP
     JENTER=J
2000 CONTINUE
C
C    NOW CALCULATE THE MAX REDUCED COST FOR SLACK VARIABLES
     N1=N+1
     NM=N+M
     DO 3000 J=N1,NM
     JD=J-N
     JJ=JBASIC(J)
     IF(JJ.GT.0) GO TO 3000
     Z=DUAL(JD)
     IF(Z.LE.IDUM) GO TO 3000
     IDUM=Z
     JENTER=J
3000 CONTINUE
     IF(IDUM.GT.0) GO TO 5000
     ISTOP=1
     IT=ITER-1
5000 CONTINUE
     RETURN
     END
C
C*********************************************************************
C    THIS SUBROUTINE CALCULATES THE UPDATED ENTERING COL
C*********************************************************************
C
     SUBROUTINE YK(Y,BASIS,A,JENTER,M,N,ISTOP,ITER)
     IMPLICIT REAL  (A-H,O-Z)
     DIMENSION Y(45),BASIS(45,45),A(45,90)
C
     IF(JENTER.GT.N) GO TO 3000
C    CASE : 1 ENTERING VAR IS NOT A SLACK VARIABLE
     DO 2000 I=1,M
     TEMP=0.
     DO 1000 J=1,M
```

```
 1000 TEMP=TEMP+BASIS(I,J)*A(J,JENTER)
      Y(I)=TEMP
 2000 CONTINUE
      GO TO 5000
C
C     CASE 2: ENTERING VARIABLE  IS A SLACK VARIABLE
 3000 CONTINUE
      J=JENTER-N
      DO 4000 I=1,M
 4000 Y(I)=BASIS(I,J)
 5000 CONTINUE
      DO 6000 I=1,M
 6000 IF(Y(I).GT.0) GO TO 9000
C     WRITE(*,7000) ITER
C7000 FORMAT(//,5X,'UNBOUNDED SOLUTION. ITER=',I4)
      ISTOP=1
 9000 CONTINUE
      RETURN
      END
C
C*********************************************************************
C     THIS SUBROUTINE DOES THE MIN RATIO TEST AND
C     UPDATES THE BASIS INVERSE,RHS AND DUALS
C*********************************************************************
C
      SUBROUTINE UPDATE(BASIS,DUAL,Y,RHS,JBASIC,IDUM,OBJ,
     *JENTER,M,N,ITER)
      IMPLICIT REAL   (A-H,O-Z)
      DIMENSION BASIS(45,45),DUAL(45),Y(45),RHS(45),JBASIC(120)
C
      RATIOM=10000000.
      DO 1000 I=1,M
      IF(Y(I).LE.0.1D-15) GO TO 1000
      RATIO=RHS(I)/Y(I)
      IF(RATIO.GE.RATIOM) GO TO 1000
      IR=I
      RATIOM=RATIO
 1000 CONTINUE
      NM=N+M
      DO 2000 J=1,NM
      IF(JBASIC(J).NE.IR) GO TO 2000
      JBASIC(JENTER)=IR
      JBASIC(J)=-IR
      GO TO 3000
 2000 CONTINUE
 3000 CONTINUE
      RHS(IR)=RHS(IR)/Y(IR)
      DO 4000 J=1,M
 4000 BASIS(IR,J)=BASIS(IR,J)/Y(IR)
      DO 6000 J=1,M
      IF(J.EQ.IR) GO TO 6000
      IF(ABS(Y(J)).LE.0.01) GO TO 6000
      RHS(J)=RHS(J)-RHS(IR)*Y(J)
C
      DO 5000 II=1,M
 5000 BASIS(J,II)=BASIS(J,II)-BASIS(IR,II)*Y(J)
 6000 CONTINUE
      DO 7000 J=1,M
 7000 DUAL(J)=DUAL(J)-IDUM*BASIS(IR,J)
      DO 8000 I=1,M
 8000 IF(RHS(I).GE.0) GO TO 9000
      WRITE(*,8200) ITER
 8200 FORMAT(//,5X,'PROBLEM IS INFEASIBLE. ITER=',I4)
```

```
 9000 CONTINUE
      OBJ=OBJ-IDUM*RHS(IR)
      RETURN
      END
C
C*********************************************************************
C     THIS SUBROUTINE PRINTS THE SOLUTION
C*********************************************************************
C
      SUBROUTINE SOLN(JBASIC,RHS,X,DUAL,OBJ,M,N)
      IMPLICIT REAL  (A-H,O-Z)
      DIMENSION JBASIC(120),RHS(45),X(90),DUAL(45)
C
      WRITE(*,4000) IT
 4000 FORMAT(/,5X,'OPTIMAL SOLUTION FOUND AT ITER=',I4)
      WRITE(*,1000) OBJ
 1000 FORMAT(/,5X,'OBJECTIVE VALUE=',F12.6)
      NM=N+M
      DO 1500 I=1,NM
 1500 X(I)=0.
      DO 2000 I=1,NM
      IF(JBASIC(I).LE.0) GO TO 2000
      TEMP=RHS(JBASIC(I))
CC    Q=Q+TEMP
      X(I)=TEMP
      WRITE(*,1200) I,X(I)
 1200 FORMAT(15X,'X(',I4,' ) = ',F12.6)
 2000 CONTINUE
C
      RETURN
      END
C
C*********************************************************************
C     THIS SUBROUTINE GENERATES RANDOM NUMBERS
C*********************************************************************
C
C     GENERATE RANDOM NUMBERS
      SUBROUTINE RAND(RAN1)
      DIMENSION IR(97)
      M=714025
      IA=1366
      IC=150889
      RM=1.0/M
      DATA IFF /0/,IDUM/-25/
      IF (IDUM.LT. 0 .OR.IFF.EQ. 0)THEN
      IFF=1
      IDUM=MOD(IC-IDUM,M)
      DO 11 J = 1,97
      IDUM = MOD(IA*IDUM+IC,M)
      IR(J)=IDUM
   11 CONTINUE
      IDUM = MOD(IA*IDUM + IC,M)
      IY = IDUM
      ENDIF
      J = 1 + (97*IY)/M
      IF(J .GT. 97.OR.J.LT.1)PAUSE
      IY = IR(J)
      RAN1 = IY*RM
      IDUM = MOD(IA*IDUM + IC,M)
      IR(J) = IDUM
      RETURN
      END
```

Appendix B.

THIS PROGRAM GENERATES AND SOLVES QUEUEING AND SCHEDULING PROBLEMS IN FORTRAN

```
C*********************************************************************
C*********************************************************************
C     THIS PROGRAM GENERATES AND SOLVES QUEUEING AND
C     SCHEDULING PROBLEMS IN FORTRAN
C
C     SOME SUBROUTINE CALLS AVAILABLE IN THE TIWAY1 HOST
C     SOFTWARE ARE USED FOR COMMUNICATION WITH THE TI530 P/C
C
C*********************************************************************
C*********************************************************************
C
C     MAIN PROGRAM FOR GENERATING THE SCHEDULING PROBLEM
C
      IMPLICIT REAL (A-H,O-Z)
      CHARACTER RI135(135)
C      INTEGER*2 RI135(135)
      INTEGER       WI4(4),WI135(135)
      INTEGER*2 RI4(4),WILEN,RILEN,NN,ISTAT,SSTAT,HWY
      INTEGER*2 XTN,RSPLEN,OUT(50)
      INTEGER*2 TAG(5),TA1(5)
      DIMENSION Q(100,100), SMIN(3), IR(97),Q1(12,10),Q2(12,10)
C      DATA M/3/, N/5/
      ISWITCH = 1
C      READ THE REQUIRED # OF QUEUES AND JOBS PER QUEUE
      READ(*,*) M,N
      RSPLEN=135
C      TAG='#0101040049'
      TAG(1)=0
      TAG(2)=1
      TAG(3)=1
      TAG(4)=03
      TAG(5)=36
      TA1(1)=0
      TA1(2)=1
      TA1(3)=1
      TA1(4)=04
      TA1(5)=133
      WILEN=1
      WI4(1)=1
      SSTAT=0
      HWY=1
      XTN=0
```

```
      ISTAT=0
      NN=1
      MM=1
      WI135(1)=1
C
      CALL INIT(ISTAT,XTN,'P1,9600,7,1,E,3,A')
C
C     THIS IS A SESSION CONTROL SUBROUTINE. IT MUST BE CALLED BY
C     ANY APPLICATION PROGRAM TO INITIATE AND TERMINATE THE USE
C     OF THE 'TIWAY' LIBRARY SUBROUTINES
C
      CALL ACTVAT(ISTAT,XTN,HWY,WILEN,WI4,RILEN,RI4)
C
C     THIS IS A HOST ADAPTER COMMAND CODE SUBROUTINE
C     BEFORE ANY COMMUNICATION IS ALLOWED WITH A SECONDARY
C     IT NEEDS TO BE LOGICALLY CONNECTED TO THE NETWORK
C     THIS IS DONE USING THE ACTVAT SUBROUTINE
C
 77   CALL TIGET(ISTAT,XTN,TAG,NN,RSPLEN,RI135,SSTAT)
C
C     TIGET IS A NIM PRIMITIVE SUBROUTINE. IT IS USED TO
C     READ CONSECUTIVE MEMORY LOCATIONS FROM A SPECIFIED
C     NIM-BASED ATTACHED DEVICE
C
      DO 20 K=1,RSPLEN
      OUT(K)= ICHAR(RI135(K))
 20   CONTINUE
      IF (OUT(1).EQ.1) THEN
          GO TO 66
        ELSE
          GO TO 77
      ENDIF
 66   DO 22 KQ = 1,3
C
C     TA1(I) IS INCREASED TO BE ABLE TO WRITE TO DIFFERENT MEMORY
C     LOCATIONS IN THE P/C WITHIN THIS DO LOOP
C
        TA1(5)=TA1(5)+2
C
      CALL QUE(M,N,Q)
C     THIS SUBROUTINE WILL SEND BACK M QUES WITH THE FIRST ELEMENT
C     BEING THE SMALLEST VALUE OF PROCESSING OF JOB I.E. SPT
      II=M/3
      JJ=(2*M)/3
      IJ = II+1
      JK = JJ+1
      SMIN(1) = 100000.
      DO 50 I = 1,II
      IF (Q(I,1).LE.SMIN(1)) SMIN(1)=Q(I,1)
 50   CONTINUE
      SMIN(2)=100000.
      DO 60 J = IJ,JJ
      IF (Q(J,1).LE.SMIN(2)) SMIN(2)=Q(J,1)
 60   CONTINUE
      SMIN(3) = 100000.
      DO 70 K = JK,M
      IF (Q(K,1).LE.SMIN(3)) SMIN(3)=Q(K,1)
 70   CONTINUE
      SPTVAL = 100000.
      DO 80 L = 1,3
      IF (SMIN(L).LE.SPTVAL) SPTVAL = SMIN(L)
 80   CONTINUE
      ISTAT=-1
```

```
      CALL TIPUT(ISTAT,XTN,TA1,MM,WI135,SSTAT)
C     WRITE(*,*) 'ISTAT=',ISTAT
C     WRITE(*,*) 'SPTVAL=',SPTVAL
 22   CONTINUE
C     DO A PCWRITE
C
C
C
      TA1(5)=133
      WI135(1)=0
      DO 32 IK=1,3
      TA1(5)=TA1(5)+2
C
      CALL TIPUT(ISTAT,XTN,TA1,MM,WI135,SSTAT)
C
C     THIS SUBROUTINE IS USED TO WRITE TO CONSECUTIVE MEMORY
C     LOCATIONS IN THE P/C
C
 32   CONTINUE
      STOP
      END
C
C*********************************************************************
C     THIS SUBROUTINE GENERATES QS AND SENDS OUT THE SMALLEST JOB
C     AS THE FIRST ELEMENT OF EACH QUEUE
C*********************************************************************
C
      SUBROUTINE QUE(M,N,Q)
      IMPLICIT REAL (A-H,O-Z)
      DIMENSION Q(100,100),Q1(10,10)
C
      DO 100 I = 1,M
C     # OF JOBS PER QUE
      DO 100 K = 1,N
      CALL RAND(RAN1)
      Q(I,K) = 10*RAN1
 100  CONTINUE
C
      CALL PENAL(M,N,Q1)
C
C     THIS ROUTINE RETURNS WITH TARDINESS PENALTY COST ASSOCIATED
C     WITH EACH JOB IN EVERY QUEUE
C
      DO 250 I=1,M
      DO 250 J = 1,N
      Q(I,J)=Q(I,J)+Q1(I,J)
 250  CONTINUE
      DO 200 I = 1,M
      DO 200 J = 1,N
      IF (Q(I,J) .LE. Q(I,1)) Q(I,1) = Q(I,J)
 200  CONTINUE
      RETURN
      END
C
C*********************************************************************
C     THIS SUBROUTINE RETURNS WITH TARDINESS PENALTY COST IN
C     TERMS OF PROCESSING TIME FOR EVERY JOB IN EVERY QUEUE
C*********************************************************************
C
      SUBROUTINE PENAL(M,N,Q1)
      IMPLICIT REAL(A-H,O-Z)
      DIMENSION Q1(10,10),Q2(10,10)
      DO 150 I= 1,M
```

```
      DO 150 K = 1,N
      CALL RAND(RAN1)
      Q2(I,K) = 10*RAN1
      Q1(I,K) = 1/(Q2(I,K))
 150  CONTINUE
      RETURN
      END
C
C*********************************************************************
C     THIS SUBROUTINE GENERATES RANDOM NUMBERS
C*********************************************************************
C
      SUBROUTINE RAND(RAN1)
      DIMENSION IR(97)
      M=714025
      IA=1366
      IC=150889
      RM=1.0/M
      DATA IFF /0/,IDUM/-25/
      IF (IDUM.LT. 0 .OR.IFF.EQ. 0)THEN
      IFF=1
      IDUM=MOD(IC-IDUM,M)
      DO 11 J = 1,97
      IDUM = MOD(IA*IDUM+IC,M)
      IR(J)=IDUM
   11 CONTINUE
      IDUM = MOD(IA*IDUM + IC,M)
      IY = IDUM
      ENDIF
      J = 1 + (97*IY)/M
      IF(J .GT. 97.OR.J.LT.1)PAUSE
      IY = IR(J)
      RAN1 = IY*RM
      IDUM = MOD(IA*IDUM + IC,M)
      IR(J) = IDUM
      RETURN
      END
```

Appendix C.

THIS PROGRAM GENERATES AND SOLVES LINEAR PROGRAMMING PROBLEMS IN BASIC ON THE NETWORK

```
1 REM*****************************************************************
2 REM*****************************************************************
3 REM THIS PROGRAM GENERATES AND SOLVES LINEAR PROGRAMMING PROBLEMS
4 REM USING THE REVISED SIMPLEX METHOD
5 REM THE PROGRAM IS WRITTEN IN BASIC AND IS COMPILED
6 REM AND RUN USING THE MICROSOFT QUICKBASIC VERSION 2.0
7 REM SOME SUBROUTINE CALLS AVAILABLE IN THE TIWAY1 HOST SOFTWARE
8 REM ARE USED FOR COMMUNICATION WITH THE TI530 PROGRAMMABLE CONTROLLER
9 REM THE LP IS OF THE FORM : MIN C*X, SUBJECT TO AX< =B, X > = 0, B> =0
10 REM M = # OF ROWS OF A, # OF CONTRAINTS
11 REM N = # OF COLUMNS OF A, # OF VARIABLES
12 REM COT(N) = COST(N),B(M), A(M,N) ARE ORIGINAL DATA
13 REM BAS(M,M)=BASIS(M,M) IS THE BASIS INVERSE
14 REM VHS(M) = RHS(M) US UPDATED RHS-VECTOR
15 REM DUL(M) IS VECTOR OF DUAL VARS;Y(M) IS UPDATED ENTERING COLUMN
16 REM JBA(M+N) IS +I IF VAR J IS BASIC AT THE ITH POSITION,-J OTHERWISE
17 REM*****************************************************************
18 DEFINT R,W,X
19 REM CHAIN MERGE "DEF.BAS", 3,ALL 'USED IN THE INTERPRETIVE MODE
20 GOSUB 65000 'NEEDED TO BE ABLE TO CALL AN EXTERNAL SUBROUTINE
21 REM
22 DEFDBL S,C,D,E,F,G,H,P,V
23 DIM BAS(45,45),A(45,90)
24 DIM DUL(45),Y(45),JBA(90),O(45),IR(97),COT(90),VHS(45),B(45)
25 DIM QUE(10),WI4(3),WI135(134),RI4(3),RI135(134),WB275(274)
26 FOR K = 0 TO 3
27 WI4(K)=0
28 RI4(K)=0
29 NEXT K
30 IST = 0
31 FOR K = 0 TO 274
32 WB275(K)=0
33 NEXT K
34 TRON
35 IT=0
36 FOR K = 0 TO 134
37 RI135(K)=0
38 WI135(K)=0
39 NEXT K
40 ITE = 1
41 WILEN = 1
42 TAG$="#0101030024"
```

```
43 WI4(0)=1
44 RI4(0)=1
45 RSTAT=0
46 SETUP$="P1,9600,7,1,E,3,A"
47 RWY=1
48 XTN=0
49 WW = 1
50 ITM = 200
51 ISW=1
52 RILEN = 1
53 IT = 0
54 RSPLEN = 135
55 TA1$="#0101040087"
56 RR=1
57 WB275(0)=1
58 WSTAT = -1 'THE SUBROUTINE BELOW IS CALLED TO INITIATE USE OF 'TIWAY
59 CALL INIT(WSTAT,XTN,SETUP$) 'LIBRARY SUBROUTINES
60 CALL ACTVAT(WSTAT,XTN,RWY,WILEN,WI4(0),RILEN,RI4(0))
61 WSTAT=-1 'THE SUBROUTINE ABOVE CONNECTS THE SECONDARY TO THE NETWORK
62 CALL TIGET(WSTAT,XTN,TAG$,WW,RSPLEN,RI135(0),RSTAT)
63 REM THE SUBROUTINE CALL ABOVE READS CONSECUTIVE MEM LOCATIONS OF P/C
64 RT1 = RSPLEN -1
65 FOR K = 0 TO RT1
66 REM QUE(K)=RI135(K)
67 NEXT K
68 IF RI135(0)< > 0 THEN GOTO 75
69 GOTO 62
70 REM
72 REM GENERATE THE INPUTS FOR THE LP PROBLEM
75 GOSUB 1000
77 REM
80 REM GET THE INITIAL BASIC FEASIBLE SOLUTION
90 GOSUB 2000
95 REM
100 REM CALCULATE THE ENTERING VARIABLE BY FINDING THE MAX Zj -Cj VALUE
110 GOSUB 3000
112 F=-25
120 IF IST < >1 THEN GOTO 150
121 REM
122 REM THIS SUBROUTINE IS USED TO WRITE TO CONSECUTIVE MEMORY LOCATIONS
124 IN A SPECIFIED NIM-BASED ATTACHED DEVICE
125 CALL TIWRIT(WSTAT,XTN,TA1$,RR,WB275(0),RSTAT)
127 REM
128 REM THIS SUBROUTINE PRINTS OUT THE SOLUTION
130 GOSUB 4000
133 WB275(0)=0
135 CALL TIWRIT(WSTAT,XTN,TA1$,RR,WB275(0),RSTAT)
140 GOTO 250
145 REM
150 REM CALCULATE THE UPDATED COLUMN OF THE ENTRING VARIAVBLE
160 GOSUB 5000
170 IF IST = 1 THEN GOTO 174 ELSE GOTO 190
171 REM ELSE GOTO 190
174 WB275(0)=1
175 CALL TIWRIT(WSTAT,XTN,TA1$,RR,WB275(0),RSTAT)
176 PRINT "ITER = ",ITE,"UNBOUNDED SOLUTION"
177 WB275(0)=0
178 CALL TIWRIT(WSTAT,XTN,TA1$,RR,WB275(0),RSTAT)
179 GOTO 250
180 REM DO THE MIN RATIO TEST. UPDATE THE BASIS INVERSE, RHS AND DUALS
190 GOSUB 6000
195 REM
200 IF ITE < ITM THEN GOTO 225
```

```
205 PRINT "TERMINATION DUE TO LIMIT ON THE # OF ITERATIONS. ITER=",ITE
210 GOSUB 4000
220 GOTO 250
225 REM CONTINUE
230 ITE = ITE+1
240 GOTO 100
250 STOP
260 END
490 REM
500 REM ***************************************************************
510 REM SUBROUTINE FOR GENERATING RANDOM NUMBERS
520 REM ***************************************************************
525 REM
530 P = 714025!
540 IZ = 1366
550 E = 150889!
560 H = 1/P
570 IF F <0 OR IT = 0 THEN GOTO 590 ELSE GOTO 730
580 REM  ELSE GOTO 730
590 IT = 1
600 S = E - F
610 GOSUB 900
620 F = FIX(D)
630 FOR U = 1 TO 97
640 S = IZ*F + E
650 GOSUB 900
660 F = FIX(D)
670 IR(U) = F
680 NEXT U
681 FOR KL=1 TO 10
683 NEXT KL
690 S = IZ * F    + E
700 GOSUB 900
710 F = FIX(D)
720 G = F
730 U = FIX(1+(97*G)/P)
740 IF U >97 OR U<1 THEN PRINT "PAUSE"
750 G = IR(U)
760 V = G*H
770 S = IZ*F+E
780 GOSUB 900
790 F = D
800 IR(U)=F
810 RETURN
820 REM
900 REM ***************************************************************
910 REM SUBROUTINE FOR CALCULATING THE MODULUS
920 REM ***************************************************************
925 REM
930 C = FIX(S/P)
935 REM PRINT "C=",C
940 D=FIX(S-(C*P))
945 REM PRINT "D=",D
950 RETURN
960 REM
1000 REM ***************************************************************
1003 REM THIS SUBROUTINE READS AND GENERATES INPUTS
1008 REM ***************************************************************
1009 REM
1015 REM THE # OF CONSTRAINTS AND VARIABLES ARE READ FROM FILE "DATA2"
1016 OPEN "DATA2" FOR INPUT AS #1
1017 REM N = 8
1018 INPUT #1, M, N
```

```
1019 IDA = 0
1020 REM READ M,N,IDA
1030 IF IDA=0 THEN GOTO 1060
1035 REM MANUALLY ENTERING VALUES FOR THE PROBLEM IS NOT ALLOWED
1040 PRINT "WRONG INPUT"
1050 GOTO 250
1055 REM
1060 REM GENERATE THE PROBLEM
1065 REM
1070 AVE = 25
1080 VHO = .6
1085 F=-25
1090 FOR JP= 1 TO N
1093 GOSUB 500
1095 VAN = V
1100 COT(JP)=AVE*VAN
1110 IF VAN > VHO THEN COT(JP)=-COT(JP)
1120 NEXT JP
1130 IF COT(3)>0 THEN COT(3)=-COT(3)
1135 REM IA = 0
1140 FOR I = 1 TO M
1143 GOSUB 500
1145 VAN = V
1150 TMP = VAN
1160 B(I)=0
1170  IF TMP> =.1 THEN B(I)=1
1180 NEXT I
1190 FOR IB= 1 TO M
1200 FOR JA = 1 TO N
1203 GOSUB 500
1205 VAN = V
1208 REM PRINT "RAN=",RAN
1210 A(IB,JA)=AVE*VAN
1220 IF VAN > VHO THEN A(IB,JA)=-A(IB,JA)
1230 NEXT JA
1240 NEXT IB
1250 NLP = N+ M
1260 N1 = N + 1
1270 FOR I = N1 TO NLP
1280  COT(I) = 0
1290 NEXT I
1300 FOR I = 1 TO M
1310 FOR J = N1 TO NLP
1320 A(I,J)=0
1330 JJ = J - N
1340 IF JJ = I THEN A(I,J) = 1
1350 NEXT J
1360 NEXT I
1370 RETURN
1380 REM
2000 REM ***********************************************************
2004 REM THIS SUBROUTINE GETS THE INITIAL BASIC FEASIBLE SOLN
2008 REM ***********************************************************
2009 REM
2010 FOR I = 1 TO M
2020 FOR J = 1 TO M
2030 BAS(I,J) = 0
2040 IF I = J THEN BAS(I,J) = 1
2050 NEXT J
2060 NEXT I
2070 FOR I = 1 TO M
2080 VHS(I) = B(I)
2090 DUL(I) = 0
```

```
2100 NEXT I
2110 N1 = N + 1
2120 NM = N + M
2130 FOR I = 1 TO N
2140 JBA(I) = -I
2150 NEXT I
2160 FOR I = N1 TO NM
2170 J = I - N
2180 JBA(I) = J
2190 NEXT I
2200 OBJ = 0
2210 RETURN
2220 REM
3000 REM ************************************************************
3003 REM THIS SUBROUTINE DETERMINES THE ENTERING VAR BY CALCULATING
3005 REM THE MAX Z(J) - C(J)
3008 REM ************************************************************
3009 REM
3010 MAX = -10000000#
3020 REM FIRST CALCULATE THE MAX RED. COST FOR REAL VARS( NON SLACKS)
3030 FOR J = 1 TO N
3040 JJ = JBA(J)
3050 IF JJ > 0 THEN GOTO 3140
3060 Z = 0
3070 FOR I = 1 TO M
3080  Z = Z+DUL(I)*A(I,J)
3085 NEXT I
3090 TMP = Z - COT(J)
3100 IF TMP < = MAX THEN GOTO 3140
3110 MAX = TMP
3120 JET = J
3140 NEXT J
3150 REM NOW CALC MAX RED COST FOR SLACK VARIABLES
3160 N1 = N + 1
3170 NM = N + M
3180 FOR J = N1 TO NM
3190 JD = J - N
3200 JJ = JBA(J)
3210 IF JJ > 0 THEN GOTO 3260
3220 Z = DUL(JD)
3230 IF Z < = MAX THEN GOTO 3260
3240 MAX = Z
3250 JET = J
3260 NEXT J
3270 IF MAX > 0 THEN GOTO 3310
3272 REM WSTAT=-1
3280 IST = 1
3290 IT = ITE - 1
3310 REM CONTINUE
3320 RETURN
3330 REM
4000 REM*************************************************************
4003 REM THIS SUBROUTINE PRINTS THE SOLUTION
4005 REM*************************************************************
4008 REM
4010 PRINT " OPTIMAL SOLUTION FOUND AT ITERATION =",IT
4018 PRINT "OBJECTIVE VALUE=",OBJ
4020 NM = N + M
4030 FOR I = 1 TO NM
4040 O(I) = 0
4050 NEXT I
4060 FOR I = 1 TO NM
4070 IF JBA(I)< = 0 THEN GOTO 4110
```

```
4080 TMP = VHS(JBA(I))
4090 O(I) = TMP
4100 PRINT "I =",I,"O(I) =",O(I)
4110 NEXT I
4120 RETURN
4130 REM
5000 REM*****************************************************************
5003 REM CALCULATE THE UPDATED ENTERING COLUMN
5005 REM*****************************************************************
5008 REM
5010 IF JET > N THEN GOTO 5110
5020 REM ENTERING VARIABLE IS NOT A SLACK VARIABLE
5030 FOR I = 1 TO M
5040 TMP = 0
5050 FOR J = 1 TO M
5060 TMP = TMP + BAS(I,J)*A(J,JET)
5070 NEXT J
5080 Y(I) = TMP
5090 NEXT I
5100 GOTO 5155
5110 REM ENTERING VAR IS A SLACK VARIABLE
5120 J = JET - N
5130 FOR I = 1 TO M
5140 Y(I) = BAS(I,J)
5150 NEXT I
5155 REM CONTINUE
5160 FOR I = 1 TO M
5170 IF Y(I) > 0 THEN GOTO 5210
5180 NEXT I
5200 IST = 1
5210  REM
5220 RETURN
5230 REM
6000 REM*****************************************************************
6003 REM THIS SUBROUTINE PERFORMS THE MIN RATIO  TEST AND UPDATES
6005 REM THE BASIS INVERSE, RHS AND DUALS
6008 REM*****************************************************************
6009 REM
6010 VAT = 100000000#
6020 FOR I = 1 TO M
6030 IF Y(I) < =1E-16 THEN GOTO 6080
6040 VTO = VHS(I)/Y(I)
6050 IF VTO > =VAT THEN GOTO 6080
6060 IR = I
6070 VAT = VTO
6080 NEXT I
6090 NM = N + M
6100 FOR J = 1 TO NM
6110 IF JBA(J) < > IR THEN GOTO 6150
6120 JBA(JET) = IR
6130 JBA(J) = -IR
6140 GOTO 6160
6150 NEXT J
6160 REM CONTINUE
6170 VHS(IR) = VHS(IR)/Y(IR)
6180 FOR J = 1 TO M
6183  BAS(IR,J) = BAS(IR,J)/Y(IR)
6185 NEXT J
6187 FOR J = 1 TO M
6190 IF J = IR THEN GOTO 6250
6200 IF ABS(Y(J)) < =.000001 THEN GOTO 6250
6210 VHS(J) = VHS(J)-VHS(IR)*Y(J)
6220 FOR II = 1 TO M
```

```
6230 BAS(J,II) = BAS(J,II)-BAS(IR,II)*Y(J)
6240 NEXT II
6250 NEXT J
6260 FOR J = 1 TO M
6270 DUL(J) = DUL(J)- MAX*BAS(IR,J)
6280 NEXT J
6290 FOR I = 1 TO M
6300 IF VHS(I) > = 0 THEN GOTO 6330
6310 NEXT I
6320 PRINT "PROBLEM IS INFEASIBLE. ITE = ",ITE
6330 REM CONTINUE
6340 OBJ = OBJ - MAX*VHS(IR)
6350 RETURN
64000 REM
64100 REM*************************************************************
64200 REM THIS SUBROUTINE CREATES AND INITIALIZES VARIABLES
64300 REM WITH SUBROUTINE NAMES TO PROPER VALUES FOR THE PURPOSE
64400 REM OF LINKING TO EXTERNAL SUNROUTINES
64500 REM*************************************************************
65000 REM
65001 REM
65002 REM
65003 REM
65004 DEF SEG = &H0
65005 I = PEEK(&H3E1)*256 + PEEK(&H3E0)
65006 DEF SEG = I
65007 TIWAY     = 0*5    'General entry point, first thing
65008 TIXTN     = 1*5    'Second half processing routine
65009 TIXTNW    = 2*5    'Second half processing routine
65010 INIT      = 3*5    'Initialization
65011 FIN       = 4*5    'Completion
65012 BRDCST    = 5*5    'Broadcast
65013 POLL      = 6*5    'Poll
65014 ACTVAT    = 7*5    'Activate secondaries
65015 DEACT     = 8*5    'Deactivate secondaries
65016 SECLOG    = 9*5    'List secondary log
65017 SDIAG     = 10*5   'Secondary diagnostics
65018 ADIAG     = 11*5   'Adapter diagnostics
65019 NATIVE    = 12*5   'Issue native task codes
65020 RDSTS     = 13*5   'Read secondary status
65021 CONFIG    = 14*5   'Read secondary configuration
65022 GETLEN    = 15*5   'Get secondary primitive sizes
65023 CHNGST    = 16*5   'Change secondary state
65024 TIREAD    = 17*5   'Read secondary memory
65025 TIWRIT    = 18*5   'Write secondary memory
65026 TIGET     = 19*5   'Read secondary memory w/conversion
65027 TIPUT     = 20*5   'Write secondary memory w/conversion
65028 FILL      = 21*5   'Fill secondary memory
65029 WRBUF     = 22*5   'Buffered write
65030 DEFBLK    = 23*5   'Define block
65031 GATHER    = 24*5   'Gather mask of data acquisition blocks
65032 WRGAT     = 25*5   'Write and gather
65033 CIMRD     = 26*5   'Read CIM memory
65034 CIMWR     = 27*5   'Write CIM memory
65035 CIMDNL    = 28*5   'Download CIM secondary
65036 CIMUPL    = 29*5   'Upload CIM secondary
65037 CCUSTS    = 30*5   'Read CIM CCU status
65038 RDLOOP    = 31*5   'Read CIM loop data
65039 RNDRD1    = 32*5   'Download random block
65040 RNDRD2    = 33*5   'Retrieve random block
65041 RNDRD3    = 34*5   'Download and retrieve random block
65042 RNDRD4    = 35*5   'Write and retrieve random block
65043 XPAR      = 36*5   'Transparent pass through
```

```
65044 LKUTGS     = 37*5    'Lookup tag - short
65045 LKUTGL     = 38*5    'Lookup tag - long
65046 LKUFMT     = 39*5    'Lookup format
65047 TI2HST     = 40*5    'TI to host format conversion
65048 HST2TI     = 41*5    'Host to TI format conversion
65049 PUTMSG     = 75*5    'Status printed on stdout
65050 GETMSG     = 76*5    'Get message for status message
65051 BLDMSK     = 77*5    'Build data acquisition mask
65052 RETURN
```

Appendix D.

THIS PROGRAM GENERATES AND SOLVES QUEUEING AND SCHEDULING PROBLEMS IN BASIC

```
1 REM*****************************************************************
2 REM*****************************************************************
3 REM THIS PROGRAM GENERATES AND SOLVES QUEUEING AND
4 REM SCHEDULING PROBLEMS. IT IS WRITTEN IN BASIC
5 REM AND IS COMPILED AND RUN USING THE MS QUICKBASIC VERSION 2.0
6 REM
7 REM SOME SUBROUTINE CALLS AVAILABLE IN THE TIWAY1 HOST SOFTWARE
8 REM ARE USED FOR COMMUNICATION WITH THE TI530 PROGRAMMABLE CONTROLLER
9 REM
10 REM****************************************************************
11 REM****************************************************************
13 DEFINT R,W,X
14 DEFDBL B,C,D,E.F,G,H,M,Z
15 REM CHAIN MERGE "DEF.BAS",20,ALL 'DEF.BAS DEFINE
16 REM THE ABOVE STATEMENT IS USED IN THE INTERPRETIVE MODE
18 GOSUB 65000
19 REM
20 REM THIS SUBROUTINE IS NECESSARY TO BRANCH TO SUBROUTINES
21 REM THAT ARE NOT CONTAINED WITHIN THE BASIC PROGRAM BEING EXECUTED
22 DIM IR(97),SMI(3),Q(150,10),ADR(2),DAT(2),SET(2),Q1(150,10),Q2(150,10)
24 DIM QUE(10)
26 DIM WI4(3),WI135(134),RI4(3),RI135(134),WB275(274)
28 DEFDBL B,C,D,E,F,G,H,M,Z
30 FOR K = 0 TO 3
32 WI4(K)=0
34 RI4(K)=0
36 NEXT K
40 FOR K = 0 TO 274
41 WB275(K)=0
43 NEXT K
44 FOR K = 0 TO 134
45 RI135(K)=0
46 WI135(K)=0
48 NEXT K
50 WILEN=1
51 TAG$="#0101030024"
52 WI4(0)=1
53 RI4(0)=1
54 RSTAT=0
55 SETUP$="P1,9600,7,1,E,3,A"
56 RWY = 1
57 XTN = 0
```

```
58 WSTAT=-1
59 WW = 1
60 ISW=1
61 RILEN = 1
63 IT = 0
65 RSPLEN=135
66 REM
70 OPEN "DATA1" FOR INPUT AS #1
71 REM
72 REM THE FILE "DATA1" CONTAINS DATA CONCERNING # OF QUEUES AND JOBS/Q
73 TA1$="#0101040087"
75 RR=1
77 WB275(0)=1
80 INPUT #1, P, N
90 F = -25
95 REM
100 CALL INIT(WSTAT,XTN,SETUP$)
102 REM
104 REM THIS IS A SESSION CONTROL SUBROUTINE. IT MUST BE CALLED
106 REM BY ANY APPLICATION PROGRAM TO INITIATE AND TERMINATE THE
108 REM USE OF THE 'TIWAY1' LIBRARY SUBROUTINES
110 CALL ACTVAT(WSTAT,XTN,RWY,WILEN,WI4(0),RILEN,RI4(0))
111 REM
112 REM THIS IS A HOST ADAPTER COMMAND CODE SUBROUTINE
113 REM IT MUST BE CALLED TO LOGICALLY CONNECT A SECONDARY TO THE
114 REM NETWORK BEFORE ANY COMMUNICATION IS ALLOWED
115 WSTAT=-1
120 CALL TIGET(WSTAT,XTN,TAG$,WW,RSPLEN,RI135(0),RSTAT)
122 REM
125 REM THIS IS A NIM-PRIMITIVE SUBROUTINE. IT IS USED TO READ
126 REM CONSECUTIVE MEMORY LOCATIONS FROM A SPECIFIED
127 REM NIM-BASED ATTACHED DEVICE
128 REM
130 RT1 = RSPLEN-1
132 FOR K = 0 TO RT1
134 QUE(K)=RI135(K)
136 NEXT K
140 IF RI135(0) < > 0 GOTO 180
150 GOTO 120
180 FOR JQ = 0 TO 2
190 REM
195 REM THREE JOBS HAVE TO BE KNOCKED OFF ONE FROM
197 REM EACH OF THREE PARTS LOADERS. HENCE THE ABOVE LOOP.
200 GOSUB 5000
210 REM THE ABOVE SUBROUTINE SENDSBACK M QUES WITH THE FIRST
220 REM ELEMENT HAVING THE SMALLEST VALUE OF PROCESSING OF EACH QUE
230 II = P/3
240 JJ = (2*P)/3
250 IJ=II+1
260 JK = JJ+1
270 SMI(1)=100000!
280 FOR I = 1 TO II
290 IF Q(I,1)< =SMI(1) THEN SMI(1)=Q(I,1)
300 NEXT I
310 SMI(2) = 100000!
320 FOR J = IJ TO JJ
330 IF Q(J,1)< =SMI(2) THEN SMI(2)=Q(J,1)
340 NEXT J
350 SMI(3) = 100000!
360 FOR K = JK TO P
370 IF Q(K,1)< =SMI(3) THEN SMI(3)=Q(K,1)
380 NEXT K
390 SPT = 100000!
```

```
400 FOR L = 1 TO 3
410 IF SMI(L) < = SPT THEN SPT = SMI(L)
420 NEXT L
430 REM PRINT "SPT=",SPT
434 IF JQ = 1 THEN TA1$="#0101040089"
435 REM
436 REM TO TURN ON THE SECOND PARTS LOADER
437 REM
438 IF JQ = 2 THEN TA1$="#010104008B"
439 REM TO TURN ON THE THIRD PARTS LOADER
440 CALL TIWRIT(WSTAT,XTN,TA1$,RR,WB275(0),RSTAT)
442 REM
444 REM THIS SUBROUTINE IS USED TO WRITE TO CONSECUTIVE
446 REM MEMORY LOCATIONS IN THE P/C
450 NEXT JQ
455 RR=10
460 TA1$="#0101040087"
463 WB275(0)=0
465 WB275(1)=0
467 WB275(2)=0
468 WB275(3)=0
469 WB275(4)=0
470 CALL TIWRIT(WSTAT,XTN,TA1$,RR,WB275(0),RSTAT)
480 STOP
490 END
500 REM ************************************************************
510 REM SUBROUTINE FOR GENERATING RANDOM NUMBERS
520 REM ************************************************************
530 M = 714025!
540 IA = 1366
550 E = 150889!
560 H = 1/M
570 IF F <0 OR IT = 0 THEN GOTO 590 ELSE GOTO 730
580 REM  ELSE GOTO 730
590 IT = 1
600 B  = E - F
610 GOSUB 1000
620 F  = FIX(D)
630 FOR J = 1 TO 97
640 B  = IA*F  +E
650 GOSUB 1000
660 F = FIX(D)
670 IR(J) = F
680 NEXT J
690 B = IA * F    + E
700 GOSUB 1000
710 F = FIX(D)
720 G = F
730 J = FIX(1 + (97*G)/M)
740 IF J >97 OR J<1 THEN PRINT "PAUSE"
750 G = IR(J)
760 Z = G * H
770 B = IA*F + E
780 GOSUB 1000
790 F = D
800 IR(J)=F
810 RETURN
820 REM
1000 REM ***********************************************************
1010 REM SUBROUTINE FOR CALCULATING THE MODULUS
1020 REM ***********************************************************
1030 C = FIX(B/M)
1040 D = FIX(B-(C*M))
```

```
1050 RETURN
5000 REM ************************************************************
5010 REM SUBROUTINE GENERATES REQUESTED # OF Q'S AND JOBS PER QUEUE
5012 REM WITH THE JOB WITH THE SHORTEST PROCESSING TIME OF EACH
5015 REM QUEUE BEING THE FIRST ELEMENT OF THAT QUEUE
5020 REM ************************************************************
5030 FOR I = 1 TO P
5040 FOR K = 1 TO N
5045 REM
5050 GOSUB 500
5055 REM CALL FOR RANDOM NUMBER
5060 MAD = Z
5070 Q(I,K) = 10 *MAD
5080 NEXT K
5090 NEXT I
5091 REM CALL FOR GETTING A TARDINESS PENALTY COST FOR EACH JOB
5092 GOSUB 6000
5093 FOR I = 1 TO P
5094 FOR K = 1 TO N
5095 Q(I,K)=Q(I,K)+Q1(I,K)
5096 NEXT K
5097 NEXT I
5100 FOR I = 1 TO P
5110 FOR J = 1 TO N
5120 IF Q(I,J)< = Q(I,1) THEN Q(I,1) = Q(I,J)
5130 NEXT J
5140 NEXT I
5150 RETURN
5160 REM
6000 REM*****************************************************************
6010 REM SUBROUTINE FOR TARDINESS PENALTY COST IN TERMS
6005 REM OF PROCESSING TIME FOR EVERY JOB IN EVERY QUEUE
6020 REM *****************************************************************
6030 FOR I = 1 TO P
6040 FOR K = 1 TO N
6050 GOSUB 500
6060 MAP = Z
6070 Q2(I,K) = 10*MAP
6080 Q1(I,K) = 1/(Q2(I,K))
6090 NEXT K
6100 NEXT I
6110 RETURN
64000 REM*****************************************************************
64500 REM THIS SUBROUTINE CREATES AND INITIALIZES VARIABLES
64550 REM WITH SUBROUTINE NAMES TO PROPER VALUES FOR THE PURPOSE
64600 REM OF LINKING TO EXTERNAL SUBROUTINES
64650 REM*****************************************************************
65000 REM
65001 REM
65002 REM
65003 REM
65004 DEF SEG=&H0
65005 I=PEEK(&H3E1)*256+PEEK(&H3E0)
65006 DEF SEG=I
65007 TIWAY      = 0*5    'General entry point, first thing
65008 TIXTN      = 1*5    'Second half processing routine
65009 TIXTNW     = 2*5    'Second half processing routine
65010 INIT       = 3*5    'Initialization
65011 FIN        = 4*5    'Completion
65012 BRDCST     = 5*5    'Broadcast
65013 POLL       = 6*5    'Poll
65014 ACTVAT     = 7*5    'Activate secondaries
65015 DEACT      = 8*5    'Deactivate secondaries
```

```
65016 SECLOG     = 9*5    'List secondary log
65017 SDIAG      = 10*5   'Secondary diagnostics
65018 ADIAG      = 11*5   'Adapter diagnostics
65019 NATIVE     = 12*5   'Issue native task codes
65020 RDSTS      = 13*5   'Read secondary status
65021 CONFIG     = 14*5   'Read secondary configuration
65022 GETLEN     = 15*5   'Get secondary primitive sizes
65023 CHNGST     = 16*5   'Change secondary state
65024 TIREAD     = 17*5   'Read secondary memory
65025 TIWRIT     = 18*5   'Write secondary memory
65026 TIGET      = 19*5   'Read secondary memory w/conversion
65027 TIPUT      = 20*5   'Write secondary memory w/conversion
65028 FILL       = 21*5   'Fill secondary memory
65029 WRBUF      = 22*5   'Buffered write
65030 DEFBLK     = 23*5   'Define block
65031 GATHER     = 24*5   'Gather mask of data acquisition blocks
65032 WRGAT      = 25*5   'Write and gather
65033 CIMRD      = 26*5   'Read CIM memory
65034 CIMWR      = 27*5   'Write CIM memory
65035 CIMDNL     = 28*5   'Download CIM secondary
65036 CIMUPL     = 29*5   'Upload CIM secondary
65037 CCUSTS     = 30*5   'Read CIM CCU status
65038 RDLOOP     = 31*5   'Read CIM loop data
65039 RNDRD1     = 32*5   'Download random block
65040 RNDRD2     = 33*5   'Retrieve random block
65041 RNDRD3     = 34*5   'Download and retrieve random block
65042 RNDRD4     = 35*5   'Write and retrieve random block
65043 XPAR       = 36*5   'Transparent pass through
65044 LKUTGS     = 37*5   'Lookup tag - short
65045 LKUTGL     = 38*5   'Lookup tag - long
65046 LKUFMT     = 39*5   'Lookup format
65047 TI2HST     = 40*5   'TI to host format conversion
65048 HST2TI     = 41*5   'Host to TI format conversion
65049 PUTMSG     = 75*5   'Status printed on stdout
65050 GETMSG     = 76*5   'Get message for status message
65051 BLDMSK     = 77*5   'Build data acquisition mask
65052 RETURN
```

Appendix E.

THIS PROGRAM GENERATES AND SOLVES LINEAR PROGRAMMING PROBLEMS IN INTERPRETED BASIC ON THE BASIC MODULE

```
1 REM ***********************************************************************
2 REM ***********************************************************************
3 REM
4 REM THIS PROGRAM GENERATES AND SOLVES LINEAR PROGRAMS
5 REM IN THE INTERPRETED BASIC ON THE BASIC MODULE
6 REM USING THE REVISED SIMPLEX METHOD
7 REM THE LP IS OF THE FORM : MIN CX, SUBJECT TO AX < = B, X> =0
8 REM                                ( B NONNEGATIVE)
9 REM
10 REM M = # OF ROWS OF A (# OF CONSTRAINTS)
11 REM N = # OF COLUMNS OF A (# OF VARIABLES)
12 REM COT(N) = COST(N), B(M), A(M,N) ARE ORIGINAL DATA
13 REM BAS(M,M) = BASIS(M,M) IS THE BASIS INVERSE
14 REM RHS(M) IS UPDATED RHS VECTOR
15 REM DUL(M) = DUAL(M) IS VECTOR OF DUAL VARIABLES
16 REM Y(M) IS UPDATED ENTERING COLUMN
17 REM JBA(M+N) = JBASIC(M+N) IS I IF VARIABLE J IS BASIC
18 REM AT THE ITH POSITION
19 REM -J OTHERWISE
21 REM
22 REM*********************************************************************
23 REM*********************************************************************
24 DIM COT[50],B[30],A[30,50],BAS[30,30],RHS[30],DAT[2]
25 DIM DUL[30],Y[30],JBA[80],X[30],IR[97],ADR[2],SET[2],VAL[2]
30 IST=0
35 IT=0
40 ITE=1
41 LET $SET[0]="Y135"
42 LET $ADR[0]="X36"
45 ID=-25
50 ITM=200
51 CALL "PCREAD",ADR[0],DAT[0],1
52 REM
53 REM THIS STATEMENT IS USED TO INVOKE SUBROUTINES TO OBTAIN
54 REM INFORMATION FROM THE P/C
55 IF DAT[0]< >0 THEN GOTO 70
57   ELSE GOTO 53
58 REM
60 REM READ IN THE INPUT DATA
70 GOSUB 1000
75 REM
80 REM GET INITIAL BASIC FEASIBLE SOLUTION
```

```
90  GOSUB 2000
95  REM
100  REM CALCULATE THE ENTERING VASRIABLE BY FINDING THE MAX ZJ-CJ VALUE
110  GOSUB 3000
120  IF IST< >1 THEN GOTO 150
122  VAL[0]=1
123  REM
125  CALL "PCWRITE",SET[0],VAL[0],1
126  REM THIS STATEMENT IS USED TO ONVOKE SUBROUTINES TO SEND
127  REM INFORMATION TO THE P/C
129  REM
130  GOSUB 4000
131  REM THIS SUBROUTINE PRINTS THE SOLUTION OF THE LP PROBLEM
132  VAL[0]=0
135  CALL "PCWRITE",SET[0],VAL[0],1
140  GOTO 250
145  REM
150  REM CALCULATE THE UPDATED COLUMN OF THE ENTERING VARIABLE
160  GOSUB 5000
170  IF IST=1 THEN GOTO 174
171    ELSE GOTO 190
174  VAL[0]=1
175  CALL "PCWRITE",SET[0],VAL[0],1
176  PRINT "ITER = ",ITE,"UNBOUNDED SOLUTION"
177  VAL[0]=0
178  CALL "PCWRITE",SET[0],VAL[0],1
179  GOTO 250
180  REM DO THE MIN RATIO TEST, UPDATE THE BASIS INVERSE,RHS AND DUALS
190  GOSUB 6000
200  IF ITE<ITM THEN GOTO 225
205  PRINT "TERMINATION DUE TO LIMIT ON THE # OF ITERATIONS. ITER=",ITE
210  GOSUB 4000
220  GOTO 250
225  REM CONTINUE
230  ITE=ITE+1
240  GOTO 100
250  STOP
260  END
1000  REM*****************************************************************
1004  REM THIS SUBROUTINE READS &  GENERATES INPUTS
1008  REM*****************************************************************
1010  DATA 8,8,0
1020  READ M,N,IDA
1030  IF IDA=0 THEN GOTO 1060
1040  PRINT "WRONG INPUT"
1050  GOTO 250
1060  REM GENERATE THE PROBLEM
1070  AVE=25
1080  RHO=0.6
1090  FOR J=1 TO N
1092   REM GET VALUES FROM THE RANDOM NO. GENERATING SUBROUTINE
1093   GOSUB 7000
1095   RAN=RAD
1100   COT[J]=AVE*RAN
1110   IF RAN>RHO THEN COT[J]=-COT[J]
1120  NEXT J
1130  IF COT[3]>0 THEN COT[3]=-COT[3]
1140  FOR I=1 TO M
1142   REM GET VALUES FROM THE RANDOM NO. GENERATING SUBROUTINE
1143   GOSUB 7000
1145   RAN=RAD
1150   TMP=RAN
1160   B[I]=0
```

```
1170  IF TMP> =0.1 THEN B[I]=1
1180 NEXT I
1190 FOR I=1 TO M
1200  FOR J=1 TO N
1201   REM GET VALUES FROM THE RANDOM NO. GENERATING SUBROUTINE
1203   GOSUB 7000
1205   RAN=RAD
1210   A[I,J]=AVE*RAN
1220   IF RAN>RHO THEN A[I,J]=-A[I,J]
1230  NEXT J
1240 NEXT I
1250 NLP=N+M
1260 N1=N+1
1270 FOR I=N1 TO NLP
1280  COT[I]=0
1290 NEXT I
1300 FOR I=1 TO M
1310  FOR J=N1 TO NLP
1320   A[I,J]=0
1330   JJ=J-N
1340   IF JJ=I THEN A[I,J]=1
1350  NEXT J
1360 NEXT I
1370 RETURN
1380 REM
2000 REM*****************************************************************
2005 REM THIS SUBROUTINE GETS      THE INITIAL BASIC FEASIBLE SOLUTION
2008 REM*****************************************************************
2009 REM
2010 FOR I=1 TO M
2020  FOR J=1 TO M
2030   BAS[I,J]=0
2040   IF I=J THEN BAS[I,J]=1
2050  NEXT J
2060 NEXT I
2070 FOR I=1 TO M
2080  RHS[I]=B[I]
2090  DUL[I]=0
2100 NEXT I
2110 N1=N+1
2120 NM=N+M
2130 FOR I=1 TO N
2140  JBA[I]=-I
2150 NEXT I
2160 FOR I=N1 TO NM
2170  J=I-N
2180  JBA[I]=J
2190 NEXT I
2200 OBJ=0
2210 RETURN
2220 REM
3000 REM*****************************************************************
3003 REM THIS SUBROUTINE
3006 REM DETERMINES THE ENTERING VAR BY CALCULATING MAX Z(J)-C(J)
3008 REM*****************************************************************
3009 REM
3010 MAX=-10000000
3020 REM FIRST CALC THE MAX RED COST FOR REAL VARS(NON SLACKS)
3030 FOR J=1 TO N
3040  JJ=JBA[J]
3050  IF JJ>0 THEN GOTO 3140
3060  Z=0
3070  FOR I=1 TO M
```

```
3080  Z=Z+DUL[I]*A[I,J]
3085  NEXT I
3090  TMP=Z-COT[J]
3100  IF TMP<=MAX THEN GOTO 3140
3110  MAX=TMP
3120  JET=J
3140 NEXT J
3150 REM NOW CALC MAX RED COST FOR SLACK VARS
3160 N1=N+1
3170 NM=N+M
3180 FOR J=N1 TO NM
3190  JD=J-N
3200  JJ=JBA[J]
3210  IF JJ>0 THEN GOTO 3260
3220  Z=DUL[JD]
3230  IF Z<=MAX THEN GOTO 3260
3240  MAX=Z
3250  JET=J
3260 NEXT J
3270 IF MAX>0 THEN GOTO 3310
3280 IST=1
3290 IT=ITE-1
3310 REM CONTINUE
3320 RETURN
3330 REM
4000 REM*****************************************************************
4002 REM THIS SUBROUTINE RPINTS THE SOLUTION
4003 REM*****************************************************************
4004 REM
4005 PRINT "OPTIMAL SLUTION FOUND AT ITERATION =",IT
4010 PRINT "OBJECTIVE VALUE =",OBJ
4020 NM=N+M
4030 FOR I=1 TO NM
4040  X[I]=0
4050 NEXT I
4060 FOR I=1 TO NM
4070  IF JBA[I]<=0 THEN GOTO 4110
4080  TMP=RHS[JBA[I]]
4090  X[I]=TMP
4100  PRINT "I=",I,"X(I)=",X[I]
4110 NEXT I
4120 RETURN
4130 REM
5000 REM*****************************************************************
5005 REM THIS SUBROUTINE  CALCULATES THE UPDATED ENTERING COLUMN
5008 REM*****************************************************************
5009 REM
5010 IF JET>N THEN GOTO 5110
5020 REM ENTERING VAR IS NOT A SLACK VARIABLE
5030 FOR I=1 TO M
5040  TMP=0
5050  FOR J=1 TO M
5060   TMP=TMP+BAS[I,J]*A[J,JET]
5070  NEXT J
5080  Y[I]=TMP
5090 NEXT I
5100 GOTO 5155
5110 REM ENTERING VAR IS A SLACK VAR
5120 J=JET-N
5130 FOR I=1 TO M
5140  Y[I]=BAS[I,J]
5150 NEXT I
5155 REM CONTINUE
```

```
5160 FOR I=1 TO M
5170  IF Y[I]>0 THEN GOTO 5210
5180 NEXT I
5200 IST=1
5220 RETURN
5230 REM
6000 REM*************************************************************** ******
6003 REM THIS SUBROUTINE
6005 REM PERFORM MIN RATION TEST AND UPDATE BASIS INVERSE, RHS AND DUALS
6007 REM*****************************************************************
6010 RAT=10000000
6020 FOR I=1 TO M
6030  IF Y[I]<=1E-16 THEN GOTO 6080
6040  RTO=RHS[I]/Y[I]
6050  IF RTO>=RAT THEN GOTO 6080
6060  IR=I
6070  RAT=RTO
6080 NEXT I
6090 NM=N+M
6100 FOR J=1 TO NM
6110  IF JBA[J]<>IR THEN GOTO 6150
6120  JBA[JET]=IR
6130  JBA[J]=-IR
6140  GOTO 6160
6150 NEXT J
6160 REM CONTINUE
6170 RHS[IR]=RHS[IR]/Y[IR]
6180 FOR J=1 TO M
6183  BAS[IR,J]=BAS[IR,J]/Y[IR]
6185 NEXT J
6187 FOR J=1 TO M
6190  IF J=IR THEN GOTO 6250
6200  IF ABS[Y[J]]<=0.000001 THEN GOTO 6250
6210  RHS[J]=RHS[J]-RHS[IR]*Y[J]
6220  FOR II=1 TO M
6230   BAS[J,II]=BAS[J,II]-BAS[IR,II]*Y[J]
6240  NEXT II
6250 NEXT J
6260 FOR J=1 TO M
6270  DUL[J]=DUL[J]-MAX*BAS[IR,J]
6280 NEXT J
6290 FOR I=1 TO M
6300  IF RHS[I]>=0 THEN GOTO 6330
6310 NEXT I
6320 PRINT "PROBLEM IS INFEASIBLE. ITER =",ITE
6330 REM CONTINUE
6340 OBJ=OBJ-MAX*RHS[IR]
6350 RETURN
6360 REM
7000 REM*****************************************************************
7005 REM THIS SUBROUTINE GENERATES RANDOM NUMBERS
7007 REM*****************************************************************
7010 P=714025
7020 IA=1366
7030 IC=150889
7040 RM=1/P
7050 IF ID<0 OR IT=0 THEN GOTO 7070
7060   ELSE GOTO 7210
7070 IT=1
7080 MO=IC-ID
7090 GOSUB 8000
7100 ID=INP[MB]
7110 FOR S=1 TO 97
```

```
7120  MO=IA*ID+IC
7130  GOSUB 8000
7140  ID=INP[MB]
7150  IR[S]=ID
7160 NEXT S
7170 MO=IA*ID+IC
7180 GOSUB 8000
7190 ID=INP[MB]
7200 IY=ID
7210 U=INP[1+(97*IY)/P]
7220 IF U>97 OR U<1 THEN PRINT "PAUSE"
7230 IY=IR[U]
7240 RAD=IY*RM
7250 MO=IA*ID+IC
7260 GOSUB 8000
7270 ID=MB
7280 IR[U]=ID
7300 RETURN
7310 REM
8000 REM*********************************************************************
8005 REM SUBROUTINE FOR CALCULATING THE MODULUS
8008 REM*********************************************************************
8010 MA=INP[MO/P]
8020 MB=INP[MO-MA*P]
8030 RETURN
```

Appendix F.

THIS PROGRAM GENERATES AND SOLVES QUEUEING AND SCHEDULING PROBLEMS IN INTERPRETED BASIC

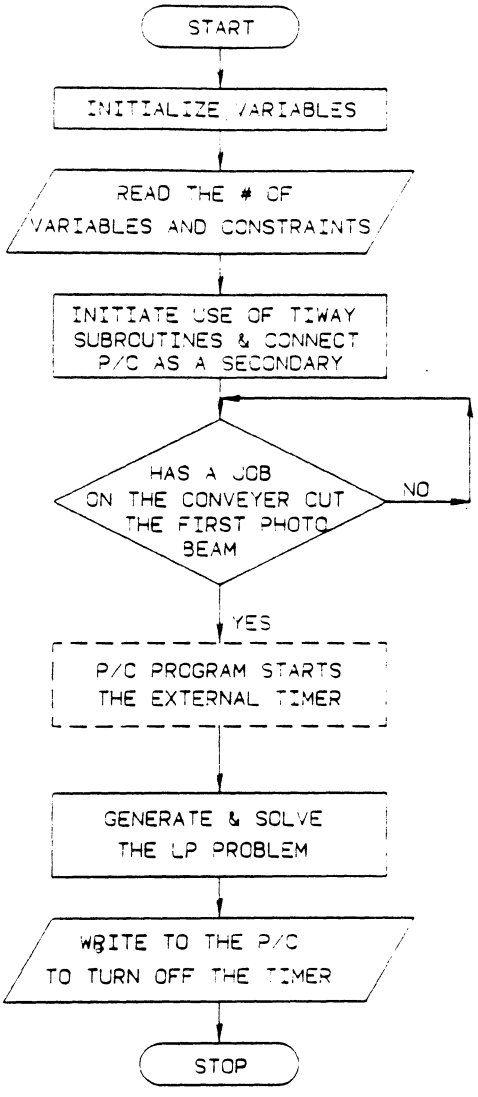
```
0  REM*****************************************************************
1  REM THIS RPOGRAM GENERATES AND SOLVES QUEUEING AND SCHEDULING
2  REM PROBLEMS IN INTERPRETED BASIC ON THE BASIC MODULE
3  REM*****************************************************************
4  DIM Q1[127,10],Q2[127,10],Q[127,10]
5  DIM IR[97],SMI[3],ADR[2],DAT[2],SET[2],VAL[2]
6  LET $ADR[0]="X36"
7  ISW=1
8  P=127  ! # OF QUES
9  ID=-25
10  N=10  ! # OF JOBS PER QUE
11  LET $SET[0]="Y135"
12  $SET[1]="Y137"
13  LET $SET[2]="Y139"
14  IT=0
15  FOR I=0 TO 2
16   DAT[I]=0
17  NEXT I
18  REM THE CALL STATEMENT BELOW IS USED TO INVOKE SUBROUTINES TO OBTAIN
19  REM INFORMATION FROM A SPECIFIC MEMORY LOCATION IN THE P/C
20  CALL "PCREAD",ADR[0],DAT[0],1
21  IF DAT[0]< >0 THEN GOTO 23
22    ELSE GOTO 20
23  FOR KQ=0 TO 2
24   REM FOR THE PURPOSE OF LOADING THREE PARTS LOADERS
25   GOSUB 500
26   REM THE ABOVE SUBROUTINE SENDS BACK M QUES WITH THE FIRST
27   REM ELEMENT HAVING THE SMALLEST VALUE OF PROCESSING OF EACH Q
28   II=P/3
30   JJ=(2*P)/3
32   IJ=II+1
34   JK=JJ+1
36   SMI[1]=100000
38   FOR I=1 TO II
40    IF Q[I,1]< =SMI[1] THEN SMI[1]=Q[I,1]
42   NEXT I
44   SMI[2]=100000
46   FOR J=IJ TO JJ
48    IF Q[J,1]< =SMI[2] THEN SMI[2]=Q[J,1]
50   NEXT J
52   SMI[3]=100000
54   FOR K=JK TO P
```

```
56   IF Q[K,1] < = SMI[3] THEN SMI[3] = Q[K,1]
58   NEXT K
60   SPT = 100000
62   FOR L = 1 TO 3
64    IF SMI[L] < = SPT THEN SPT = SMI[L]
66   NEXT L
69   REM      PRINT "SPT = ",SPT
70   VAL[KQ] = 1  ! SET Y51 TO ON
71   REM THE CALL STATEMENT IS USED TO INVOKE SUBROUTINES THAT
72   REM SEND INFORMATION TO THE P/C
73   CALL "PCWRITE",SET[KQ],VAL[KQ],1
74  NEXT KQ
75  FOR KQ = 0 TO 2
76   VAL[KQ] = 0
77   CALL "PCWRITE",SET[KQ],VAL[KQ],1
79  NEXT KQ
80  STOP
90  END
100  REM*****************************************************************
101  REM THIS SUBROUTINE RETURNS A RANDOM NUMBER
102  REM*****************************************************************
104  M = 714025
105  IA = 1366
107  IC = 150889
108  RM = 1/M
130  IF ID < 0 OR IT = 0 THEN GOTO 142
135    ELSE GOTO 220
142  IT = 1
143  MO = IC-ID
145  GOSUB 300
147  ID = INP[MB]
160  FOR J = 1 TO 97
163   MO = IA*ID + IC
165   GOSUB 300
170   ID = INP[MB]
180   IR[J] = ID
190  NEXT J
193  MO = IA*ID + IC
195  GOSUB 300
200  ID = INP[MB]
210  IY = ID
220  J = INP[1 + (97*IY)/M]
230  IF J > 97 OR J < 1 THEN PRINT "PAUSE"
240  IY = IR[J]
250  RAN = IY*RM
253  MO = IA*ID + IC
255  GOSUB 300
260  ID = MB
270  IR[J] = ID
280  RETURN
300  REM*****************************************************************
303  REM THIS SUBROUTINE IS USEDFOR CALCULATING THE MODULUS
305  REM*****************************************************************
310  MA = INP[MO/M]
320  MB = INP[MO-MA*M]
330  RETURN
500  REM*****************************************************************
503  REM THIS SUBROUTINE RETURNS REQUESTED # OF QS AND JOBS PER QUEUE
504  REM WITH THE FIRST JOB IN EACH QUEUE BEING THE ONE WITH
506  REM THE SHORTEST PROCESSING TIME
508  REM*****************************************************************
510  FOR I = 1 TO P
520   FOR K = 1 TO N
```

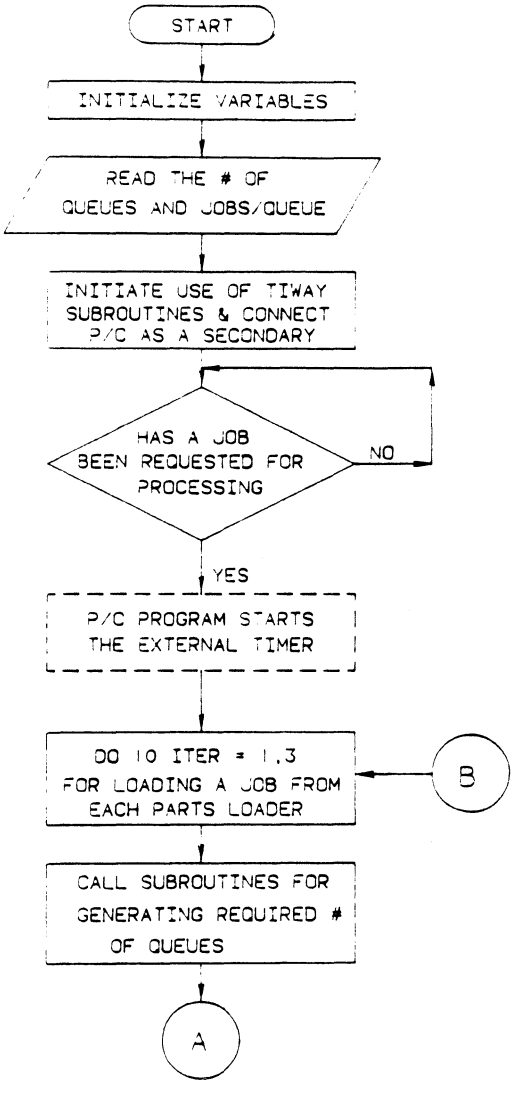
```
524   GOSUB 100
528   RAD=RAN
530   Q[I,K]=10*RAD
540  NEXT K
550 NEXT I
552 GOSUB 800
553 FOR I=1 TO P
554  FOR J=1 TO N
555   Q[I,J]=Q[I,J]+Q1[I,J]
556  NEXT J
557 NEXT I
560 FOR I=1 TO P
570  FOR J=1 TO N
580   IF Q[I,J]< =Q[I,1] THEN Q[I,1]=Q[I,J]
590  NEXT J
600 NEXT I
610 RETURN
800 REM ***************************************************************
810 REM THIS SUBROUTINE RETURNS WITH A PENALTY COST IN TERMS
814 REM OF PROCESSING TIME FOR EACH JOB IN EACH QUEUE
820 REM ***************************************************************
830 FOR I=1 TO P
840  FOR K=1 TO N
850   GOSUB 100
860   RAP=RAN
870   Q2[I,K]=10*RAP
880   Q1[I,K]=1/(Q2[I,K])
890  NEXT K
900 NEXT I
910 RETURN
```

Appendix G.

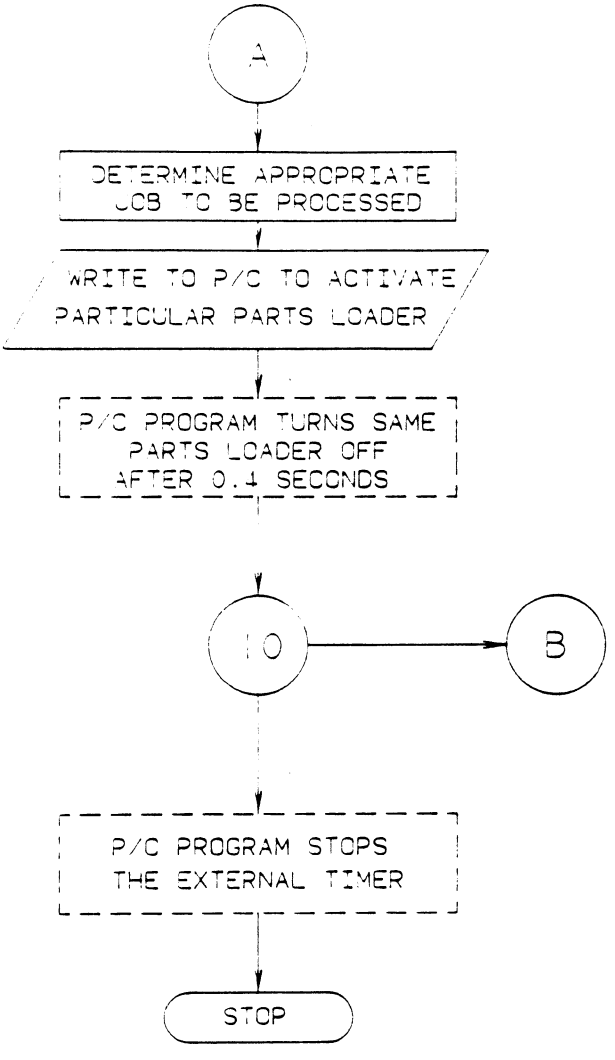
FLOW CHARTS FOR ALGORITHMS DEVELOPED FOR COMPUTATIONALLY INTENSIVE TASKS



Flowchart for Linear Programming Task



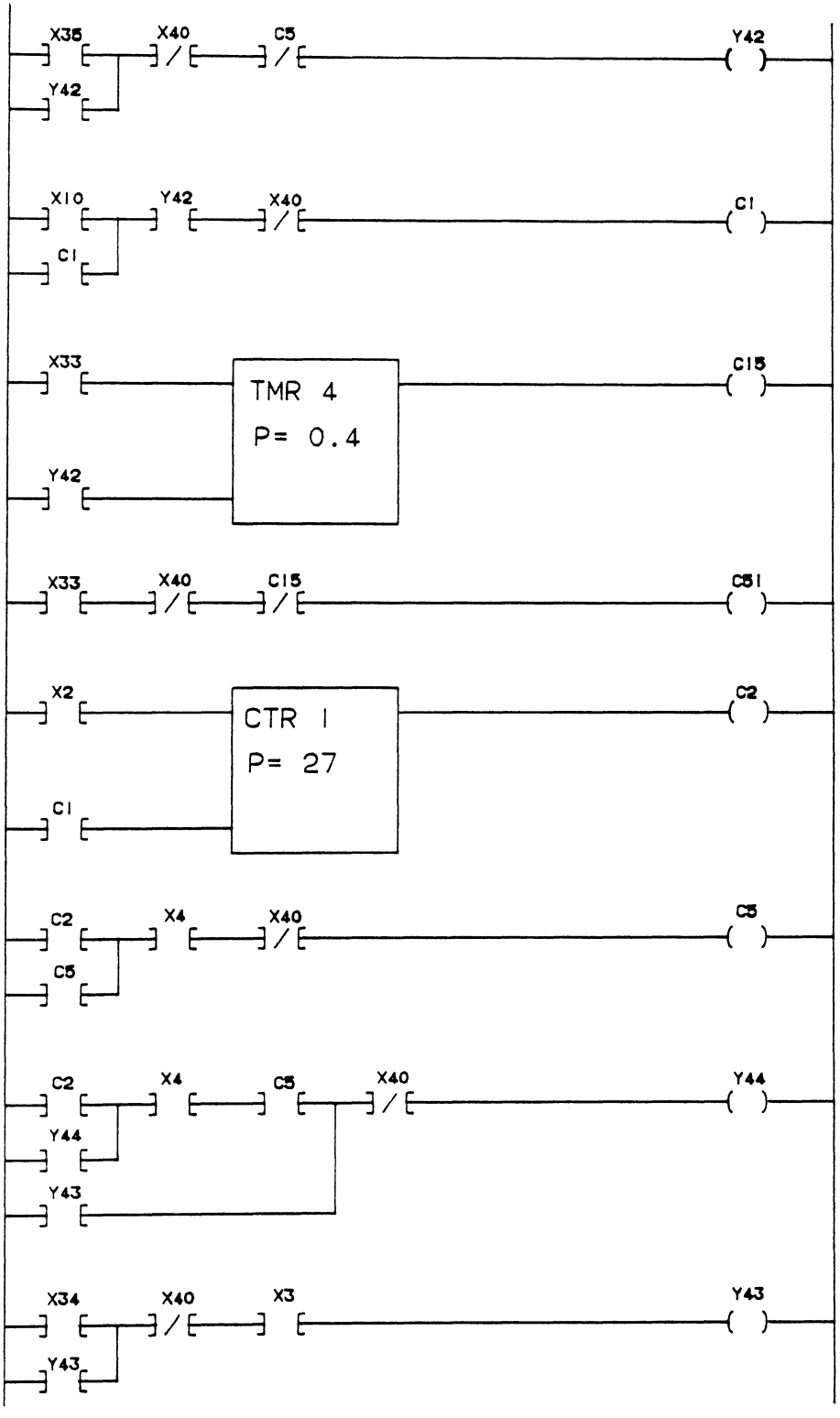
Flowchart for Queueing and Scheduling Task

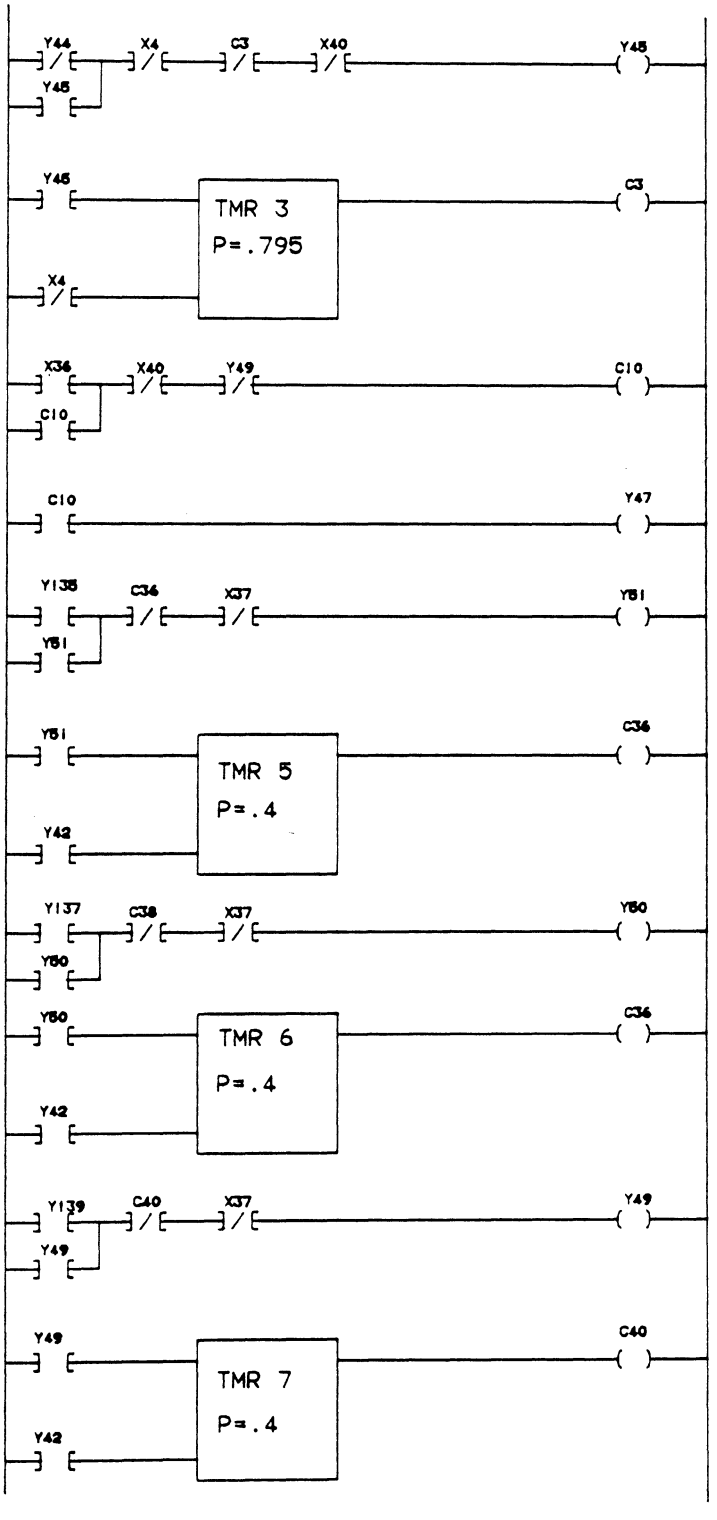


Flowchart for Queueing and Scheduling Task

Appendix H.

RELAY LADDER LOGIC PROGRAM USED IN TESTS


X35
X40
C5
Y42
Y42
X10
Y42
X40
C1
C1
X33
TMR 4
P= 0.4
C15
Y42
X33
X40
C15
C51
X2
CTR 1
P= 27
C2
C1
C2
X4
X40
C5
C5
C2
X4
C5
X40
Y44
Y44
Y43
X34
X40
X3
Y43
Y43


Y44
X4
C3
X40
Y45
Y45
Y45
TMR 3
P=.795
C3
X4
X36
X40
Y49
C10
C10
C10
Y47
Y135
C36
X37
Y51
Y51
Y51
TMR 5
P=.4
C36
Y42
Y137
C38
X37
Y50
Y50
Y50
TMR 6
P=.4
C36
Y42
Y139
C40
X37
Y49
Y49
Y49
TMR 7
P=.4
C40
Y42

Appendix I.

THIS PROGRAM WAS USED FOR PERFORMING DATA TRANSFER TESTS IN THE BASIC MODULE

```
2 REM*****************************************************************
4 REM*****************************************************************
5 REM
6 REM  THIS PROGRAMIS RUN ON THE BASIC MODULE
8 REM  IT IS USED FOR PERFORMING TESTS FOR DATA TRANSFER
10 REM CAPABILITIES BETWEEN THE PROGRAMMABLE CONTROLLER
12 REM AND THE BASIC MODULE FOR NON CONSECUTIVE DISCRETE
14 REM AND WORD MEMORY LOCATIONS
16 REM
18 REM*****************************************************************
20 REM*****************************************************************
30 DIM ADR[1000],DAT[1000],SET[2],VAL[3],SAF[1],SAG[1],SAH[1]
31 DIM SEX[1],SEY[1],SEZ[1],SAA[1],SAB[1],SAC[1],SAD[1],SAE[1]
32 DIM SEA[1],SEB[1],SEC[1],SED[1],SEE[1],SEF[1],SEG[1],SEH[1]
33 DIM SEI[1],SEJ[1],SEK[1],SEL[1],SEM[1],SEN[1],SEO[1]
34 DIM SEP[1],SEQ[1],SER[1],SES[1],SEU[1],SEV[1],SEW[1]
35 LET $SET[0]="Y47"
36 VAL[0]=1
37 REM
38 REM THE FOLLOWING CHARACTER VARIABLES DEFINE DIFFERENT
39 REM NON CONSECUTIVE WORD AND DISCRETE DATA POINTS IN THE P/C
40 LET $ADR[0]="V01"
41 LET $SEX[0]="V03"
42 LET $SEY[0]="C01"
43 LET $SEZ[0]="X08"
44 LET $SAA[0]="Y10"
45 LET $SAB[0]="V12"
46 LET $SAC[0]="TCP1"
47 LET $SAD[0]="WX1"
48 LET $SAE[0]="Y15"
49 LET $SEA[0]="V15"
50 LET $SEB[0]="C10"
51 LET $SEC[0]="X25"
52 LET $SED[0]="Y05"
53 LET $SEE[0]="X45"
54 LET $SEF[0]="WX10"
55 LET $SEG[0]="X55"
56 LET $SEH[0]="Y19"
57 LET $SEI[0]="C100"
58 LET $SEJ[0]="Y33"
59 LET $SEK[0]="X47"
60 LET $SEL[0]="C50"
```

```
61  LET $SEM[0]="WY5"
62  LET $SEN[0]="TCC5"
63  LET $SEO[0]="TCP6"
64  LET $SEP[0]="X48"
65  LET $SEQ[0]="V55"
66  LET $SER[0]="X49"
67  LET $SES[0]="X53"
68  LET $SEU[0]="C66"
69  LET $SEV[0]="Y42"
70  LET $SEW[0]="Y47"
71  LET $SAF[0]="Y48"
72  LET $SAG[0]="Y49"
73  LET $SAH[0]="Y50"
74  FOR I=0 TO 99
75   DAT[I]=0
80  NEXT I
85  REM
90  CALL "PCWRITE",SET[0],VAL[0],1
95  REM
100  REM THIS STATEMENT IS USED TO INVOKE SUBROUTINES THAT WRITE TO
110  REM MEMORY LOCATIONS IN THE P/C. HERE IT IS USED TO TURN ON THE
115  REM EXTERNAL TIMER
120  REM
141  CALL "PCREAD",SEX[0],DAT[2],1
142  CALL "PCREAD",SEY[0],DAT[4],1
143  CALL "PCREAD",SEZ[0],DAT[6],1
145  CALL "PCREAD",SAA[0],DAT[8],1
150  CALL "PCREAD",ADR[0],DAT[0],1
151  CALL "PCREAD",SAB[0],DAT[10],1
154  CALL "PCREAD",SAC[0],DAT[12],1
156  CALL "PCREAD",SAD[0],DAT[14],1
158  CALL "PCREAD",SAE[0],DAT[16],1
160  CALL "PCREAD",SAF[0],DAT[18],1
162  CALL "PCREAD",SAH[0],DAT[20],1
164  CALL "PCREAD",SAG[0],DAT[22],1
166  CALL "PCREAD",SEA[0],DAT[24],1
168  CALL "PCREAD",SEB[0],DAT[26],1
170  CALL "PCREAD",SEC[0],DAT[28],1
172  CALL "PCREAD",SED[0],DAT[30],1
174  CALL "PCREAD",SEE[0],DAT[32],1
175  CALL "PCREAD",SEF[0],DAT[34],1
176  CALL "PCREAD",SEG[0],DAT[36],1
178  CALL "PCREAD",SEH[0],DAT[38],1
180  CALL "PCREAD",SEI[0],DAT[40],1
182  CALL "PCREAD",SEJ[0],DAT[42],1
184  CALL "PCREAD",SEK[0],DAT[44],1
186  CALL "PCREAD",SEL[0],DAT[46],1
188  CALL "PCREAD",SEM[0],DAT[48],1
190  CALL "PCREAD",SEN[0],DAT[50],1
192  CALL "PCREAD",SEO[0],DAT[52],1
194  CALL "PCREAD",SEP[0],DAT[54],1
196  CALL "PCREAD",SEQ[0],DAT[56],1
198  CALL "PCREAD",SER[0],DAT[58],1
200  CALL "PCREAD",SES[0],DAT[60],1
202  CALL "PCREAD",SEU[0],DAT[62],1
203  REM
204  REM THE "PCREAD" STATEMENTS ARE USED TO INVOKE SUBROUTINES THAT
205  REM OBTAIN INFORMATION FROM THE P/C. THERE ARE 32 DIFFERENT
206  REM "PCREAD" STATEMENTS TO TEST FOR DATA TRANNSFER CAPABILITY
207  REM FOR 32      NON-CONSECUTIVE DISCRETE AND WORD DATA POINTS
208  REM
212  REM      CALL "PCREAD",SEV[0],DAT[64],750
214  REM
```

```
216 REM THIS STATEMENT WAS USED TO TEST FOR DATA TRANSFER CAPABILITY
217 REM FOR CONSECUTIVE DISCRETE AND WORD DATA POINTS
219 VAL[0]=0
220 CALL "PCWRITE",SET[0],VAL[0],1
225 REM
230 REM THIS CALL STATEMENT IS USED TO TURN OFF THE EXTERNAL TIMER
240 STOP
250 END
```

Appendix J.

THIS PROGRAM PERFORMS TESTS FOR DATA TRANSFER CAPABILITIES BETWEEN THE P/C AND THE HOST FOR CONSECUTIVE DATA POINTS

```
C*********************************************************************
C*********************************************************************
C
C     THIS PROGRAM WAS USED FOR PERFORMING TESTS FOR DATA TRANSFER
C     CAPABILITIES BETWEEN THE P/C AND THE HOST COMPUTER
C     FOR CONSECUTIVE DISCRETE AND WORD DATA POINTS
C
C*********************************************************************
C*********************************************************************
C
      CHARACTER RSP275(275),RSP1(275),RSP2(275),RSP3(275),RSP4(275)
      CHARACTER RSP5(275),RSP6(275),RSP7(275)
      INTEGER      WI4(4),WB275(275),NN,OUT(275)
      INTEGER*2 RI4(4),WILEN,RILEN,ISTAT,HWY,RI135(135)
      INTEGER*2 XTN,RSPLEN,LIST(32),WI135(135)
      INTEGER*2 TAG(5),TA1(5),SSTAT
      CHARACTER PORTSTR*20,TAG32*20
      CHARACTER TAG20*20,TAG21*20,TAG22*20,TAG23*20,TAG24*20,TAG25*20
      CHARACTER TAG26*20,TAG27*20,TAG28*20,TAG29*20,TAG30*20,TAG31*20
C     TAG24 = '#0101060001'
      TAG25 = '#0101010001'
      TAG26 = '#0101010083'
      TAG27 = '#0101010105'
      TAG28 = '#0101010187'
      TAG29 = '#0101010209'
      TAG30 = '#010101021B'
      TAG31 = '#010101030D'
      TAG32 = '#010101038F'
      RSPLEN = 275
      PORTSTR = 'P1,9600,7,1,E,3,A'
      TAG(1) = 0
      TAG(2) = 1
      TAG(3) = 1
      TAG(4) = 01
      TAG(5) = 01
      TA1(1) = 0
      TA1(2) = 1
      TA1(3) = 1
      TA1(4) = 04
      TA1(5) = 47
      WILEN = 1
      WI4(1) = 1
```

```
      SSTAT = 0
      HWY = 1
      XTN = 0
      ISTAT = 0
      NN = 1
      MM = 1
      M25 = 130
      M26 = 130
      M27 = 135
      M28 = 130
      M29 = 130
      M30 = 130
      M31 = 130
      M32 = 130
      WI135(1) = 1
C
      CALL INIT(ISTAT,XTN,PORTSTR)
C
C     THIS IS A SESSION CONTROL SUBROUTINE AND IT MUST BE CALLED
C     BY ANY APPLICATION PROGRAM TO INITIATE AND TERMINATE THE
C     USE OF THE 'TIWAY' LIBRARY SUBROUTINES
C
      CALL ACTVAT(ISTAT,XTN,HWY,WILEN,WI4,RILEN,RI4)
C
C     THIS IS A HOST COMMAND CODE SUBROUTINE
C     BEFORE ANY COMMUNICATION IS ALLOWED WITH A SECONDARY
C     IT MUST BE LOGICALLY CONNECTED TO THE NETWORK
C
      ISTAT = -1
      RSPLEN = 275
C
C     READ(*,*) M1
C     CALL TIREAD(ISTAT,XTN,TAG24,M1,RSPLEN,RSP1,SSTAT)
C
C     THIS SUBROUTINE WAS USED TO READ CONSECUTIVE DISCRETE DATA
C     POINTS. THE TAG24 ARGUMENT SPECIFIES READING OF THE CONSECUTIVE
C     DISCRETE DATA POINTS IN A COMPACT FORMAT(I.E. ONE DATA POINT
C     FOR EVERY BIT).
C     M1 WAS USED TO INPUT THE NUMBER OF DATA POINTS NEEDED TO BE
C     ACCESSED
C
      CALL TIPUT(ISTAT,XTN,TA1,NN,WI135,SSTAT)
C     THIS SUBROUTINE IS USED TO WRITE TO MEMORY LOCATIONS IN THE
C     P/C. IN THIS CASE IT WRITES TO START THE TIMER
C
C     THE FOLLOWING TIREAD STATEMENTS WERE USED TO BE ABLE TO
C     READ UPTO 960 WORD MEMORY LOCATIONS SINCE
C     EACH TIREAD STATEMENT CAN READ ONLY 134
C     CONSECUTIVE WORD LOCATIONS
C
      ISTAT = -1
      CALL TIREAD(ISTAT,XTN,TAG27,M27,RSPLEN,RSP3,SSTAT)
      CALL TIREAD(ISTAT,XTN,TAG28,M28,RSPLEN,RSP4,SSTAT)
      CALL TIREAD(ISTAT,XTN,TAG29,M29,RSPLEN,RSP5,SSTAT)
      CALL TIREAD(ISTAT,XTN,TAG30,M30,RSPLEN,RSP6,SSTAT)
      CALL TIREAD(ISTAT,XTN,TAG31,M31,RSPLEN,RSP7,SSTAT)
C
      WI135(1) = 0
      CALL TIPUT(ISTAT,XTN,TA1,NN,WI135,SSTAT)
C
C     THIS CALL WAS USED TO TURN OFF THE TIMER
      STOP
      END
```

Appendix K.

THIS PROGRAM PERFORMS TESTS FOR DATA TRANSFER CAPABILITIES BETWEEN THE P/C AND THE HOST FOR NON CONSECUTIVE DATA POINTS

```
C***********************************************************************
C***********************************************************************
C
C     THIS PROGRAM TESTS THE DATA TRANSFER CAPABILITY BETWEEN
C     THE PROGRAMMABLE CONTROLLER AND THE HOST COMPUTER ON THE
C     TIWAY1 NETWORK USING THE GATHER SUBROUTINE CALL
C     AVAILABLE IN THE TIHOST SOFTWARE.  THIS SUBROUTINE CALL
C     CAN ACCESS UPTO 32 DIFFERENT NON-CONSECUTIVE WORD OR DISCRETE
C     MEMORY LOCATIONS THRUOGH ONE CALL
C
C     THIS PROGRAM IS WRITTEN IN FORTRAN
C
C***********************************************************************
C***********************************************************************
      CHARACTER RSP275(275)
      INTEGER*4 TAGLST(32)
C      INTEGER*2 TAGLST(32)
      INTEGER       WI4(4),WB275(275),NN,OUT(275)
      INTEGER*2 RI4(4),WILEN,RILEN,ISTAT,HWY,RI135(135)
      INTEGER*2 XTN,RSPLEN,LIST(32),WI135(135)
      INTEGER*4 MASK
      INTEGER*2 TAG(5),TA1(5),NBLK
      INTEGER*2 SSTAT,NNNNLST(32),CCLST(32)
      CHARACTER PORTSTR*20,TAGS(32)*20,TA2*20,TAG7*20
      RSPLEN = 270
      NBLK = 32
      TA2 = '#0101'
      PORTSTR = 'P1,9600,7,1,E,3,A'
      TAG(1) = 0
      TAG(2) = 1
      TAG(3) = 1
      TAG(4) = 01
      TAG(5) = 01
      TA1(1) = 0
      TA1(2) = 1
      TA1(3) = 1
      TA1(4) = 04
      TA1(5) = 47
      WILEN = 1
      WI4(1) = 1
      SSTAT = 0
      HWY = 1
```

```
      XTN = 0
      ISTAT = 0
      NN = 1
      MM = 1
      WI135(1) = 1
   59 CONTINUE
C
C     THE CHARACTER VARIABLES BELOW INDICATE THE START OF ADDRESS OF
C     THE LOCATION WHICH NEEDS TO BE ACCESSED BY THE GATHER CALL
C
      TAGS(1) = '#0101030021'
      TAGS(2) = '#0101010064'
      TAGS(3) = '#0101010138'
      TAGS(4) = '#0101040021'
      TAGS(5) = '#0101030064'
      TAGS(6) = '#0101010238'
      TAGS(7) = '#0101030044'
      TAGS(8) = '#0101010164'
      TAGS(9) = '#010101013D'
      TAGS(10) = '#0101030021'
      TAGS(11) = '#0101010074'
      TAGS(12) = '#0101040148'
      TAGS(13) = '#0101030011'
      TAGS(14) = '#0101010066'
      TAGS(15) = '#0101030133'
      TAGS(16) = '#0101030022'
      TAGS(17) = '#0101010069'
      TAGS(18) = '#0101010188'
      TAGS(19) = '#0101030031'
      TAGS(20) = '#0101010094'
      TAGS(21) = '#0101010148'
      TAGS(22) = '#0101030022'
      TAGS(23) = '#0101010065'
      TAGS(24) = '#01010101A8'
      TAGS(25) = '#0101030061'
      TAGS(26) = '#0101040164'
      TAGS(27) = '#0101040138'
      TAGS(28) = '#0101030069'
      TAGS(29) = '#0101010033'
      TAGS(30) = '#0101010199'
      TAGS(31) = '#0101030046'
      TAGS(32) = '#0101010464'
      DO 88 I = 1,32
      TAGLST(I) = LOCFAR(TAGS(I))
   88 CONTINUE
      DO 69 I = 1,32
      CCLST(I) = 0
   69 CONTINUE
      DO 37 I = 1,32
      CCLST(I) = I
   37 CONTINUE
      DO 79 I = 1,32
      NNNNLST(I) = 0
   79 CONTINUE
      DO 47 I = 1,32
      NNNNLST(I) = 1
   47 CONTINUE
      DO 89 I = 1,32
      LIST(I) = 0
   89 CONTINUE
C
      CALL INIT(ISTAT,XTN,PORTSTR)
C
```

```
C     THIS IS A SESSION CONTROL SUBROUTINE. IT MUST BE CALLED
C     BY ANY APPLICATION PROGRAM TO INITIATE AND TERMINATE
C     THE USE OF 'TIWAY' LIBRARY SUBROUTINES
C
      CALL ACTVAT(ISTAT,XTN,HWY,WILEN,WI4,RILEN,RI4)
C
C     THIS SUBROUTINE IS FOR LOGICALLY CONNECTING A SECONDARY
C     TO THE NETWORK
C
      ISTAT = -1
C
      CALL DEFBLK(ISTAT,XTN,TA2,NBLK,CCLST,TAGLST,NNNNLST,SSTAT)
C
C     THIS SUBROUTINE CALL IS USED TO DEFINE P/C MEMORY LOCATIONS
C     WHICH SUBSEQUENTLY CAN BE CALLED BY THE GATHER CALL
C
      DO 57 I = 1,32
      LIST(I) = I
   57 CONTINUE
C
      CALL BLDMSK(MASK,LIST)
C
C     THIS SUBROUTINE CALL BUILDS A MASK FROM A LIST OF BLOCK NUMBERS
C     THIS MASK IS AN ARGUMENT WHICH THE CALL GATHER USES
C
      ISTAT = -1
C
      CALL TIPUT(ISTAT,XTN,TA1,NN,WI135,SSTAT)
C
C     THIS CALL IS USED TO WRITE TO CONSECUTIVE MEMORY LOCATIONS
C     IN A SPECIFIED NIM-BASED DEVICE. IN THIS EXPERIMENT THE TIMER
C     IS STARTED AT THIS POINT
C
      CALL GATHER(ISTAT,XTN,TA2,MASK,RSPLEN,RSP275,SSTAT)
C
C     THIS SUBROUTINE CALL IS USED TO GATHER DAT A FORM PREVIOUSLY
C     DEFINED DATA ACQUISITION BLOCKS. DATA FROM MULTIPLE BLOCKS
C     CAN BE RETURNED IN THE SAME CALL
C
      WI135(1) = 0
      CALL TIPUT(ISTAT,XTN,TA1,NN,WI135,SSTAT)
C
C     THE TIMER IS PUT OFF AT THIS POINT, IMMEDIATELY AFTER THE GATHER
C     CALL
C
      DO 120 K = 1,RSPLEN
      OUT(K) = ICHAR(RSP275(K))
  120 CONTINUE
      STOP
      END
```

Appendix L.

TABULATED VALUES OF RESPONSE TIMES FOR ALL TESTS PERFORMED

Response time in seconds for consecutive word data points

Number of data points	Int. BASIC on BASIC M	FORTRAN on TIWAY
30	0.10	0.47
60	0.15	0.76
90	0.21	0.99
120	0.26	1.25
150	0.31	1.57
180	0.36	1.85
210	0.41	2.22
240	0.46	2.35
480	0.87	4.42
960	1.70	8.92

Response time in seconds for consecutive discrete data points

Number of data points	Int. BASIC on BASIC M	Fortran on TIWAY
26	0.10	0.25
52	0.15	0.26
78	0.20	0.29
104	0.26	0.30
208	0.48	0.35
316	0.74	0.43
448	1.0	0.48
896	1.92	0.71

Response time in seconds for non-consecutive data points

Number of data points	Int. BASIC on BASIC M	TIREAD on TIWAY	GATHER on TIWAY
1	0.10	0.22	0.26
2	0.15	0.42	0.33
3	0.19	0.54	0.45
4	0.24	0.67	0.47
5	0.28	0.85	0.66
10	0.53	1.55	1.04
20	1.01	2.94	1.79
32	1.58	4.62	3.3

Response time in seconds for Queueing and Scheduling task

Number of queues	Number of jobs/que	Int BASIC on BASIC M	Fortran on TIWAY	Comp QBASIC on TIWAY	Int BASIC on TIWAY
5	10	12.87	1.15	5.76	77.86
10	10	24.27	1.83	10.42	144.86
25	10	58.41	2.87	24.40	345.83
50	10	115.34	4.75	47.58	680.50
75	10	172.08	6.46	70.83	
100	10	229.11	8.22	94.04	
127	10	290.50	10.05	119.18	

Response times in seconds for the Linear Programming task

Number of constraints	Number of variables	Number of iterations	Int. BASIC on BASIC M	Fortran on TIWAY	Comp QBASIC on TIWAY	Int. BASIC on TIWAY
5	5	3	4.0	.45	2.55	29.98
10	10	7	13.62	1.27	8.18	100.16
15	15	8	29.0	2.58	17.45	216.34
20	20	8	48.51	4.3	29.30	362.95
25	25	13	102.63	9.3	64.17	778.13
32	40	12	171.55	16.4	108.6	

The vita has been removed from the scanned document