

**DEVELOPMENT OF A MONITORING AND DATA LOGGING SYSTEM  
FOR A MULTI-LINE TELEPHONE CONSOLE**

**By Andras Mathys Zsigmond Molnar**

Thesis submitted in partial fulfilment of the requirements for the Masters Diploma in Technology to the Department of Electrical Engineering (light current) at the Cape Technikon.

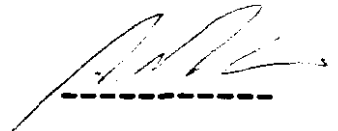
Research and Development Centre  
Department of Posts and Telecommunications

CAPE TOWN  
SOUTH AFRICA  
NOVEMBER, 1991

**DECLARATION**

I declare that the contents of this thesis represents my own work and the opinions contained herein are my own. It has not been submitted before for any examination at this or any other institute.

**A. M. Z. MOLNAR**

A handwritten signature in cursive script, appearing to read 'A. M. Z. Molnar', is written over a horizontal dashed line.

**(Signature)**

## **ABSTRACT**

This thesis describes the design, development and implementation of a monitoring and data logging system for a multi-line telephone console as required by the Account Enquiry section of the Department of Posts and Telecommunications.

## **OPSOMMING**

Hierdie verhandeling beskryf die ontwerp, ontwikkeling en uitvoering van 'n monitering en data versameling fasiliteit vir 'n multi-lyn telefoon konsole soos benodig deur die Rekening Navrae seksie van die Departement van Pos en Telekommunikasiewese.

## ACKNOWLEDGEMENTS

I would sincerely like to extend a special word of thanks to Mr D Nel for his encouragement and assistance throughout the duration of this project.

I would especially like to thank Mr J P Calitz for his patience and friendly advice in helping with the software development and for his constructive criticism regarding my design ideas.

I would earnestly like to thank Mr M Amerika for always being prepared to be of assistance and for his ingenious ideas regarding hardware design in particular, which was an enormous help during this project.

I would like to thank Mr A Herbert and his staff at the Research and Development Centre of the Department of Posts and Telecommunications for all the help received during this project.

I would also like to thank Plessey for being prepared to grant me access to their library and for all their assistance regarding the BTS 60 system.

## TABLE OF CONTENTS

	Page
1. INTRODUCTION.	1-1
1.1 The Existing Account Enquiry Section.	1-1
1.2 The Solution.	1-3
2. OBJECTIVE.	2-1
3. INFORMATION SYSTEM DEVELOPMENT.	3-1
3.1 The Feasibility Study.	3-1
3.2 The System Analysis.	3-2
3.2.1 Analysis of the Present System.	3-2
3.2.2 Analysis of the System Requirements.	3-2
3.3 System Design.	3-3
3.3.1 Logical System Design.	3-3
3.3.2 Physical System Design.	3-4
3.3.3 System Specifications.	3-5
3.4 Software Development.	3-5
3.5 System Implementation.	3-6
3.5.1 Training.	3-6
3.5.2 Testing.	3-7
3.5.3 Documentation.	3-7
3.6 System Maintenance.	3-8
3.7 Summary.	3-8
4. HARDWARE DESIGN.	4-1
4.1 The Visual Display.	4-1
4.2 The Power Supply.	4-2

4.3	The Ringing Detection Circuits.	4-2
4.4	Multiplex and Count the Ringing Calls.	4-3
4.5	Latched and Decoder Driven Seven-Segment Displays.	4-4
4.6	Maintenance and Error Checking.	4-5
4.7	PC Board Schematic Drawings.	4-6
5.	SOFTWARE DEVELOPMENT.	5-1
5.1	The Pascal Language.	5-1
5.1.1	Why TURBO Pascal?	5-2
5.2	Data Capturing, Logging and Monitoring System.	5-2
5.3	Welcome to Monitor!	5-3
5.3.1	The Main Menu.	5-5
5.3.2	The Set-up Utility.	5-7
5.3.2.1	The Communications Port.	5-8
5.3.2.2	The Baud Rate.	5-9
5.3.2.3	The Parity.	5-10
5.3.2.4	The Data Bits.	5-11
5.3.2.5	The Stop Bits.	5-12
5.3.2.6	The Input Buffer Size.	5-13
5.3.2.7	The Output Buffer Size.	5-14
5.3.2.8	Do Xon/Xoff.	5-15
5.3.2.9	Do Clear to Send.	5-16
5.3.2.10	Do Data Set Ready.	5-17
5.3.3	Print Reports.	5-18
5.3.3.1	Daily Switchboard Statistics.	5-19
5.3.3.2	Individual Extensions.	5-20

5.3.3.3	Trunk Line Reports.	5-21
5.3.3.4	Average Answering Times.	5-22
5.3.3.5	All Reports.	5-22
5.3.3.6	Printer Error.	5-23
5.3.3.7	Processing Data for Printed Report.	5-24
5.3.4	System Date and Time.	5-25
5.3.5	Office Lay-out and Extension Numbers.	5-29
5.3.6	Quit.	5-30
5.4	Archiving.	5-30
5.5	Handshaking.	5-30
5.6	Program Flowcharts and Listings.	5-31
6.	PROBLEMS ENCOUNTERED.	6-1
7.	FUTURE ENHANCEMENTS.	7-1
7.1	Dynamically Changing the Software.	7-1
7.2	Monthly and Quarterly Generated Reporting.	7-1
7.3	Graphical Representation of Information.	7-2
7.4	Dynamic Report Generation.	7-3
8.	CONCLUSION.	8-1
9.	BIBLIOGRAPHY.	9-1
APPENDIX A.	Schematic Drawings.	A-1
APPENDIX B.	Flowcharts.	B-1
APPENDIX C.	Program Listings.	C-1

## **1 INTRODUCTION.**

The Department of Posts and Telecommunications Commercial Account Enquiry section has a run-of-the-mill management problem. This problem can be described as the lack of information required by management for effective decision-making and control. To emphasize the situation, one should take a look at an overview of the existing Account Enquiry section.

### **1.1 The Existing Account Enquiry Section.**

The Account Enquiry section provides the telephone using public with an information service. This service includes information such as telephone account information, telephone suspension and restoration statistics as well as various general queries related to the national and international telephone networks.

The section consists of managerial, supervisory and operational staff as well as operational hardware. The operational hardware consists of a business telephone system (BTS 60) and various computer terminals that are utilized to obtain relevant account and subscriber information. The operational staff comprises a number of operators who answer the incoming BTS switchboard.



## INTRODUCTION

The control and effective utilization of the operators, switchboard and section as a whole was an almost impossible task due to the lack of relevant information regarding the performance of the operators and the switchboard. The only information available to management regarding the performance of the operators and the switchboard was a logged print-out of all the calls passing through the BTS 60 and lists of calls attended to by the individual operators. The calls passing through the system are logged as they occur and are not sorted in any specific manner. This information is of little or no value to management as the logged print-out offers reams of information that need to be transformed into a meaningful format. The list of calls attended to is recorded by the operators themselves and thus the integrity of this information is debatable.

Due to the increasing number of complaints received from the private sector regarding the accessibility of the Account Enquiry service and the section's own desire to improve its public image it became apparent that management information was imperative. With the aid of such information it would be possible to manage and control the section in a more efficient and organized manner, thus providing an improved service.

**1.2 The Solution.**

A possible solution to the Account Enquiry section's problem would be to design some sort of management information system (MIS).

This management information system would be used to collect, transform and transmit the necessary information required by the controlling staff. With the aid of such management information, the Account Enquiry section would be able to operate more efficiently as well as provide a better service to the public sector. Consequently the section's image and that of the Department of Posts and Telecommunications could be improved considerably.

## **2 OBJECTIVE.**

The main objective of this project is to design an effective management information system that will present the Account Enquiry section with the necessary information to establish an integrated management system.

Of course, even the best information cannot guarantee good decisions if managers do not have the ability to use it effectively. This is why the information must be presented in a form which is easy to understand and use. Managerial and supervisory staff require information of high quality and not of great quantity. It must be emphasized that managers cannot possibly absorb all of the information that can be produced by information systems. Thus, high-quality information must possess several major characteristics in order to effectively support managerial decision-making. For example, information should generally have the following qualities:

- Timely. Information must be available when needed.
- Accurate. Information must be free from errors.
- Complete. All information needed must be provided.
- Concise. Only information needed must be provided.

## OBJECTIVE

In conjunction with the above information criteria an effort must be made to establish the nature of the required information needed to suit the strategy and functional plan of the section. These information requirements will be obtained by means of a feasibility study.

On completion of the feasibility study a system analysis must be conducted before the system can be designed. Once the design is complete the software can be developed to meet the specifications of the design stage. After this all that remains is the implementation of the system.

### **3 INFORMATION SYSTEM DEVELOPMENT.**

The development cycle for this management information system includes various stages viz. a feasibility study, system analysis, system design, software development, implementation and maintenance of the designed system. The activities involved are highly related and interdependent.

#### **3.1 The Feasibility Study.**

A feasibility study is a preliminary study that determines the information needs of prospective users and the objectives, constraints, basic resource requirements and feasibility of a proposed project.

The information needed for such a feasibility study and for the other stages of the system development was gathered from various sources, viz. (1) personal interviews with users, operators and managers, (2) personal observation of the system in action, (3) the examination of documents, reports, data media, procedural manuals and other systems documentation. The information gathered in this manner was used for the preliminary specifications of the proposed system, including the system criteria and constraints.

### **3.2 The System Analysis.**

The final product of system analysis is the system requirements or functional specifications for the proposed information system.

#### **3.2.1 Analysis of the Present System.**

To determine these system requirements and functional specifications, one has to gain some sort of knowledge regarding the organizational environment in which the system is to be located.

This entails learning more about the Account Enquiry section as an organization, its management structure, its business activities and the information system it has installed. Once the organizational environment has been scrutinized, the actual system requirements can be dealt with.

#### **3.2.2 Analysis of the System Requirements.**

The focus of the system requirements analysis must be on the information requirements of the prospective users of the new system and the logical input/output, processing, storage and control requirements of the proposed system.

## INFORMATION SYSTEM DEVELOPMENT

This is the most difficult step of system analysis. Firstly one must determine what the users specific information needs are. Secondly one must determine the information processing capabilities needed for each system function (input, processing, output, storage and control) to meet these information requirements. Finally one should develop the logical system requirements. This means that one should not tie your analysis to the physical resources of hardware, software and people that might be used. That's a job for the system design stage.

### **3.3 System Design.**

System design involves developing both a logical and physical design for an information system that meets the system requirements developed in the system analysis stage.

#### **3.3.1 Logical System Design.**

Logical system design involves developing general specifications for how the basic information processing system functions of input, processing, output, storage and control can meet the user's requirements.

## INFORMATION SYSTEM DEVELOPMENT

Logical system design should also include the consideration of an ideal system and alternative realistic systems. Developing general specifications for an ideal system encourages the users and designer to be creative and emphasizes the fact that meeting the information requirements of the section is the primary goal of the project. The development of an alternative realistic system encourages the users and designer to be flexible and realistic and emphasizes the fact that a solution must be found that meets the requirements, while taking into account the limited financial, personnel and other resources of the section.

### **3.3.2 Physical System Design.**

Physical system design involves the detailed design of input/output methods and media, data bases and processing procedures. Users and designers use their knowledge of the section's operations, information processing and computer hardware and software to develop the physical design of the system. They must relate their design to the input, processing, output, storage and control functions of the proposed system. They must specify what types of hardware, software and people resources will be needed to transform the data resources into an information product.



### **3.3.3 System Specifications.**

The final step of the system design stage is the development of the system specifications. It includes specifications for source documents, the database and output media, procedures for data preparation and collection and information processing procedures. It also includes specifications for the hardware and software that will be used by the new system.

### **3.4 Software Development.**

The software development stage involves the programming process through which computer programs are developed that meet the specifications of the design stage. It must be emphasized that the programming stage requires continual interaction between the users, system analyst, system designer and the programmer. The system specifications and system design developed may be continually refined and revised during the programming , implementation and maintenance stages of the system development.

### **3.5 System Implementation.**

The activities of system implementation involve the testing, documenting, installing and operation of the newly designed system and the training of personnel to operate and use the system.

#### **3.5.1 Training.**

Implementation of a new system involves orientation and training of management, users and operating personnel and the "selling" of the new system to each of these groups. Users and operating personnel must be trained in specific skills to operate and use the system. If management and the relevant users have been adequately involved in the system development stage, the shock effect of transferring to a new system should be minimized. If user representatives participate in the development of the new system the problems of installation, conversion and training should be minimized.

**3.5.2 Testing.**

System implementation requires the testing of the newly designed and programmed system. This involves not only the testing and debugging of all computer programs, but the testing of all other data processing procedures, including the production of test copies of reports and other output that should be reviewed by the users of the proposed system for possible errors. Testing not only occurs during the system's implementation stage, but throughout the system's development process.

**3.5.3 Documentation.**

System documentation is an important process that uses the tools and techniques of system analysis and design to record and communicate the activities and results of each stage of the system development. Proper documentation allows management to monitor the progress of the project and to minimize the problems that arise when changes are made in the system design. It is also vital for proper implementation and maintenance since installing and operating a newly designed system requires a detailed record of the system design.

### **3.6 System Maintenance.**

The system maintenance activity requires a periodic review or audit of the system to ensure that it is operating properly and meeting its design objectives. This activity is in addition to the continual monitoring of the new system. System maintenance also includes making modifications to the system due to changes within the section or environment.

### **3.7 Summary.**

Due to the size and nature of this project the feasibility study, system analysis, system design, software development and implementation of the designed management information system was conducted by the author. The views expressed in this chapter were taken into consideration and employed in the development of this project. A description of the hardware and software development will be pursued in the ensuing chapters.

#### **4 HARDWARE DESIGN.**

In conjunction with the management information requirements, the Account Enquiry section also requires a visual display that can indicate the amount of incoming calls waiting to be answered. The reason for such a display is to indicate, to the operators and supervisors, at what rate the switchboard is being accessed. This can then be utilized as a motivational tool to increase the efficiency with which the staff handle the incoming calls to the switchboard.

##### **4.1 The Visual Display.**

The display is driven via hardware directly and interfaced to the incoming telephone lines on a mini-distribution frame. The incoming lines on the distribution frame are physically connected to ringing detection circuits. These circuits monitor the lines for ringing current and if detected they output a particular potential. All the outputs from the detection circuits are taken via a multiplexer and counting circuitry to establish how many lines are ringing. The number of lines ringing are then decoded and taken via device drivers to the seven-segment displays.

#### 4.2 The Power Supply.

The power supply, designed to supply the rest of the hardware circuitry with the necessary power to function correctly, is a very basic yet efficient supply. It is a simple transformer-bridge-capacitor-regulator power supply that converts the 220V ac voltage to a +5V and +12V dc voltage.

#### 4.3 The Ringing Detection Circuits.

In designing these circuits, it is of the utmost importance to abide by the Department of Posts and Telecommunications regulations (SAPO SPEC 3B24/001) regarding interfacing of external equipment to the physical telephone lines.

These specifications state, in paragraph 4.3.1, that a circuit for detecting incoming ringing shall preferably be connected in parallel across the telephone line. The circuit shall be capacitively coupled. The value of capacitance as seen from the line shall be nominally 1,8 to 2,2 micro-farads. The detecting circuit impedance, including the capacitor, at 17 Hertz and 25 Hertz shall be between 4000 ohms and 10 000 ohms when tested at 35V and 75V RMS.

The constraints set for the capacitor value and the impedance is to ensure that the device, during the line testing from an exchange, yields similar conditions to a SAPO telephone. This, in turn, ensures uniformity of line conditions for all installations.

The ringing detection circuits are designed to detect ringing directly from the incoming telephone lines that service the switchboard and thus the above specifications as laid down by the Department of Posts and Telecommunications are strictly adhered to in the design of the ringing detection circuits.

The Account Enquiry switchboard has 24 incoming lines and thus 24 ringing detection circuits were required. Each of these 24 circuits have an output that indicates whether the attached incoming line is ringing. These outputs are then taken via a multiplexer to counting circuitry.

#### 4.4 Multiplexing and Counting the Ringing Calls.

The multiplexer is used to scan the ringing detection circuits' outputs individually and step the relevant counters when ringing is detected.

The multiplexer's inputs are connected to the ringing detection circuits and its output is connected to the various counters' clocks. Each individual input is selected cyclically via binary controlled inputs on the multiplexer. Thus, for each input on the multiplexer that detects ringing, the relevant counters are clocked. After all the inputs have been scanned in this manner the values of the counters are passed to the seven-segment displays via latching and decoder driven circuits. The counters are then reset and the cycle starts again.

#### 4.5 Latched and Decoder Driven Seven-Segment Displays.

Latch decoder drivers are used to interface the BCD outputs of the counters to the seven-segment displays.

The latch decoder drivers used for this project have lamp test, blanking and latch enable or strobe inputs which are provided to test the displays, shut off or intensity-modulate it and store or strobe the BCD codes. These functions are needed to blank the tens digit on the displays when there are no incoming lines ringing and to store the BCD codes for latching so as to prevent the displays from flashing in conjunction with the incoming ringing tone.



The displays that are used for this project are high-intensity common-cathode seven-segment displays. Their physical dimensions are such, that all the operators and supervisors manning the Account Enquiry switchboard can comfortably read the information displayed on them.

#### 4.6 Maintenance and Error Checking.

The manner in which this visual display was designed and constructed is the culmination of a variety of factors. Primary factors that contributed to this design are practical considerations such as the ease of maintenance and error checking.

The consideration of providing easier maintenance and error checking go hand in hand. This is accomplished by having each separate stage, such as the power, ringing detection circuits, etc. as an individual module. The physical cabling between such modules are not soldered directly to the PC boards, but are instead connected via plug-in connectors for easy disconnection when localizing faults. The integrated circuits and displays are mounted by means of sockets instead of directly soldering them to the PC boards which enables them to be inserted and removed with ease.

The various PC boards, displays, transformer and cabling are all housed in a single perspex container. This container comprises of a base plate with a box-like lid. The lid can also be slid off the base plate so as to make access to the PC boards and accessories as convenient as possible.

#### 4.7 PC Board Schematic Drawings.

The schematic drawings of each individual module used in the design of this visual display is detailed in Appendix A.

## **5 SOFTWARE DEVELOPMENT.**

The software development was done with the aid of the TURBO Pascal language as implemented for the MS-DOS operating system.

### **5.1 The Pascal Language.**

Pascal is a general-purpose, high level programming language originally designed by Professor Niklaus Wirth of the Technical University of Zurich, Switzerland and named in honour of Blaise Pascal, the famous Seventeenth Century French philosopher and mathematician.

Professor Wirth's definition of the Pascal language, published in 1971, was intended to aid the teaching of a systematic approach to computer programming, specifically introducing structured programming. TURBO Pascal is designed to meet the requirements of all categories of users. It offers the student a friendly interactive environment which greatly aids the learning process and in the hands of a programmer it becomes an extremely effective development tool providing both compilation and execution times second to none.

### 5.1.1 Why TURBO Pascal?

TURBO Pascal was decided on, due to its systematic approach to computer programming and modular structure.

It is this systematic approach and modular structure that contributes to an extremely effective development tool. This development tool simplifies the writing, debugging and understanding of the programming sequence. The execution time of any high level programming language can, however, hardly be compared to that of a low level language such as assembler, which is extremely fast. Due to the nature of this specific application, the execution time of the specific software is not a major problem. TURBO Pascal, in conjunction with the above mentioned facts also presents a friendly interactive environment to the programmer.

### 5.2 Data Capturing, Logging and Monitoring System.

The software developed specifically for this application is called Monitor. Monitor consists of a number of modules that include communication, data capturing, file management, data processing, screen and keyboard handling and reporting.

The communication module takes care of initializing the PC's RS-232 serial port to match the parameters of the local BTS 60 system. On completion, the RS-232 serial port is then ready to receive data sent to the PC from the BTS 60 system. Data that is received is then captured to a disk file on the PC's hard drive. This file is then manipulated to extract the necessary information needed for the screen and reporting facilities. The screen is used, with the aid of various menu's, to present to the end user the available information and facilities. These facilities, such as setting up various communication parameters, selecting pre-designed reports, checking or updating the system time and date or viewing the office layout, can be selected by means of the keyboard.

### **5.3 Welcome to Monitor!**

Monitor was designed to be user friendly, user tolerant and to validate data entered by the user from the keyboard. Special consideration was also given to making the program simple to operate for both novice and advanced computer operators. The ease of use will be indicated when discussing the various menu's.

The benefits of using menu's in the Monitor application are as follows:

- displays all the choices available
- reduces the amount of technical knowledge required
- adds structure to the application
- makes it easier to learn, use and maintain
- minimizes typing errors
- reduces training and support costs.

The diagram in Fig. 5.0 is a menu tree showing the relationships among the menu's and menu items of the Monitor application.

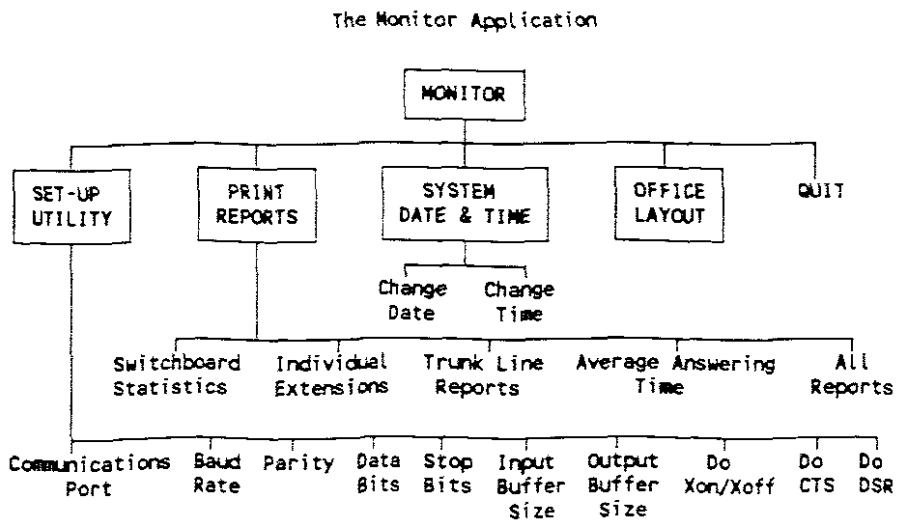


Fig 5.0

5.3.1 The Main Menu.

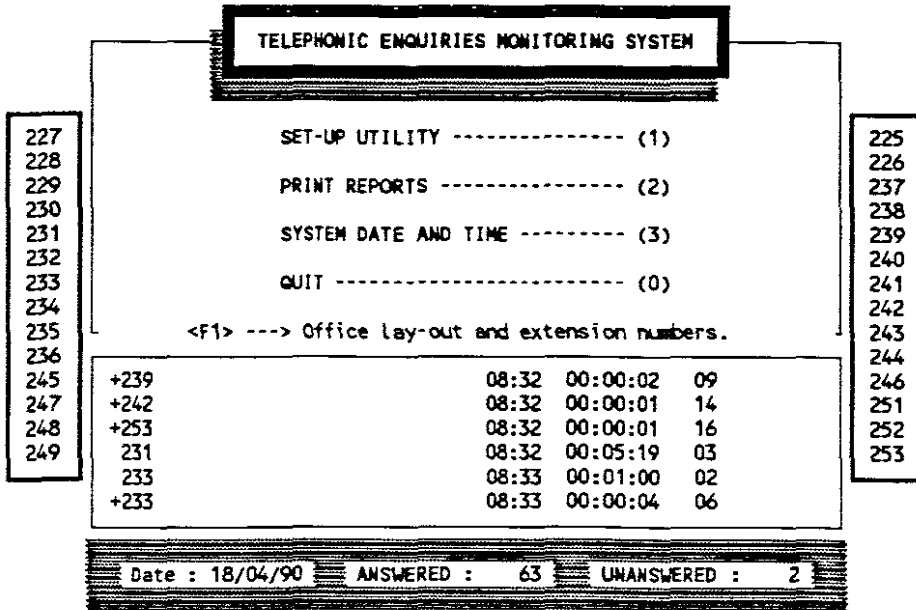


Fig. 5.1

The Main Menu (See Fig. 5.1) was designed to be visually stimulating as well as informative. The menu offers the user real-time data, real-time information and the opportunity to select various other menu's. To select any of the menu's or options, simply press the related key indicated between brackets.

The main block, or be it the top block on the main menu, presents the user with the opportunity of selecting between the Office Layout, Set-up Utility, Print Reports and the System Date and Time menu's. It also has an option to Quit the program and return to the operating system.

The window directly below this block displays to the user real-time data as it is received from the BTS.

The block at the bottom of the menu displays the current date and information regarding the amount of calls answered and unanswered by the switchboard on a real-time basis.

The two blocks on the left and right-hand side of the menu are a visual indication of the various extension numbers available on the switchboard. The numbers pertaining to extensions that are busy, are displayed in reverse video. When an extension returns to normal (i.e. the call has been serviced) the corresponding number is again displayed in normal video. This facility allows the user to monitor the switchboard as a whole indicating which extensions are busy or dormant at any specific time.



## 5.3.2 The Set-up Utility.

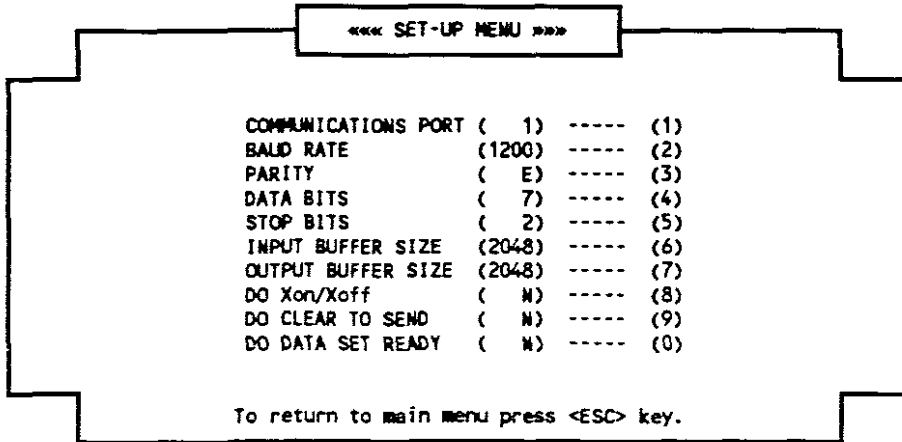


Fig. 5.2

The Set-up menu was designed as a means for the user to set up communication parameters. These parameters include selecting the required communications port, baud rate, parity, number of data bits, number of stop bits, size of input buffer, size of output buffer and whether to do Xon/Xoff, clear to send or data set ready checking.

The settings in brackets immediately after the various menu options, indicate the default parameters for these options. To change any of these parameters, simply press the related key indicated between the second set of brackets.

## 5.3.2.1 The Communications Port.

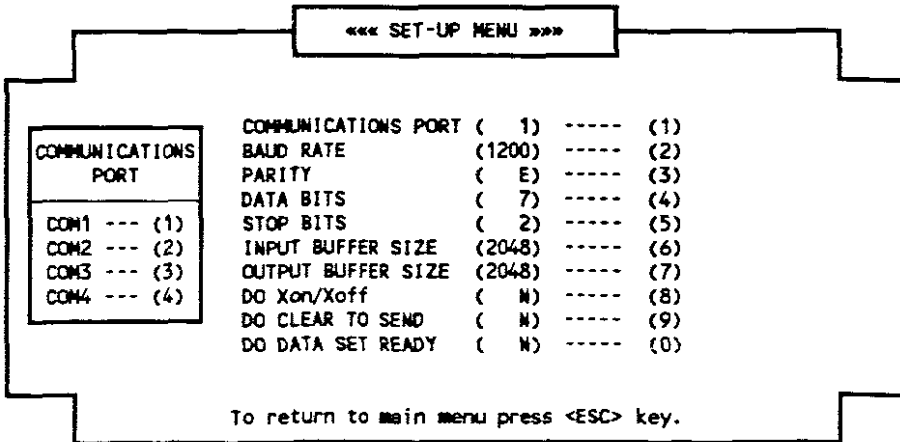


Fig. 5.2.1

If option (1) of the Set-up Utility is requested a pop-up menu appears. This allows the user to select which communications port is to be used to communicate with the BTS 60. To select, for example, the first Asynchronous Communications Adapter (ACA) port, option (1) on the pop-up menu should be selected.

When sending data from the PC to the BTS, the ACA will translate that data into a form which can be interpreted by the BTS 60. Likewise, when data is transmitted from the BTS 60 to the PC, the ACA translates the incoming data into a format that can be understood by the PC.

## 5.3.2.2 The Baud Rate.

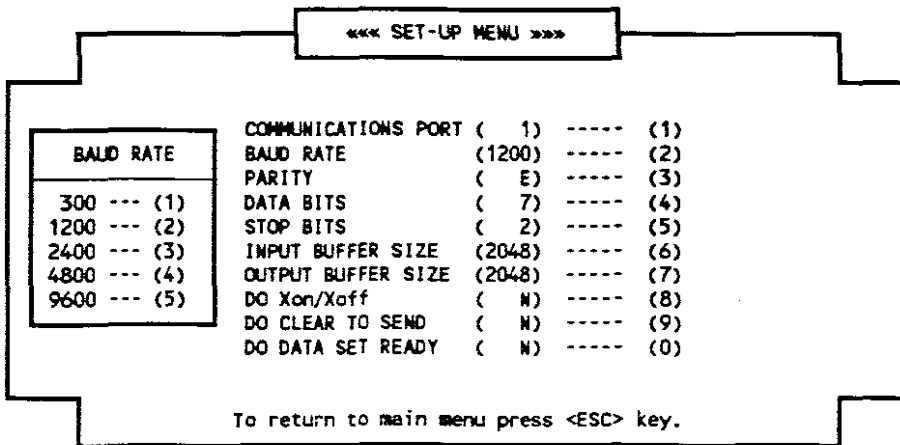


Fig. 5.2.2

If option (2) of the Set-up Utility is requested a pop-up Baud Rate menu appears. This parameter specifies the speed at which the data is transferred between the PC and the BTS 60. The speed is measured in bits per second which generally ranges from 300 to 9600 baud.

Because PC's have a wide range of operating speeds, it must be ensured that the data transmission speed of the BTS 60 and the PC are equal. Failure to match the baud rate will result in garbled data.

## 5.3.2.3 The Parity.

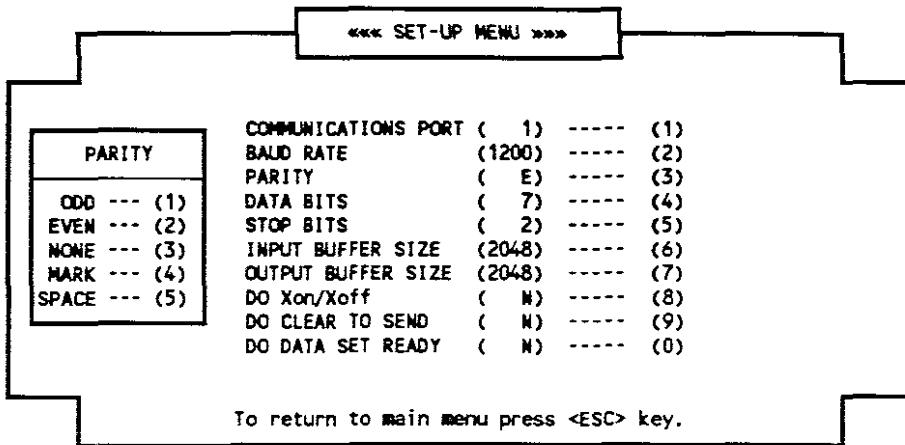


Fig. 5.2.3

If option (3) of the Set-up Utility is requested a pop-up Parity menu appears. This menu allows the user to specify the character parity to be used on the currently selected port.

The parity technique involves adding a bit, called a parity bit, to a data word. A parity bit may take the value of 1 or 0 depending on the number 1's in the data word and the employed parity scheme. Parity checking is a commonly used method for checking data integrity.

## 5.3.2.4 The Data Bits.

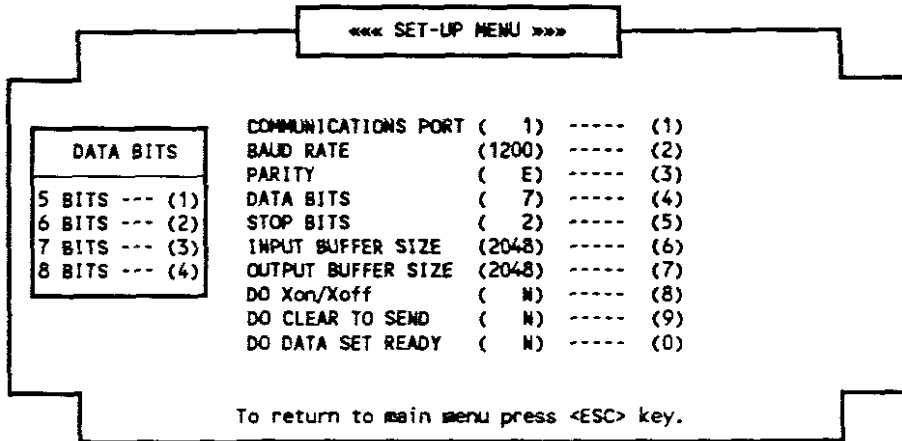


Fig. 5.2.4

If option (4) of the Set-up Utility is requested a pop-up Data Bit's menu appears. This allows the user to specify the number of bits that constitutes each character of data.

While the PC supports word lengths of 5,6,7, or 8 bits, the most commonly used word lengths are 7 and 8 bits. Parity checking is normally enabled for 7-bit transmission and disabled for 8-bit. This gives us a consistent word size of 8 bits for each byte of data transmitted.

## 5.3.2.5 The Stop Bits.

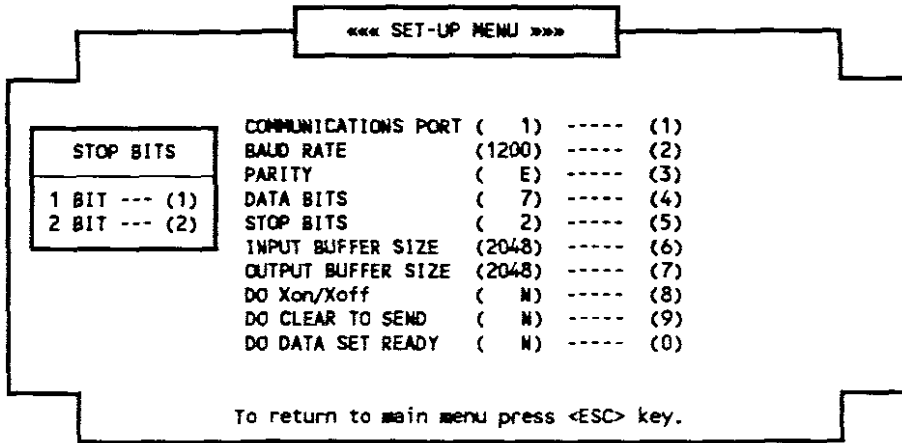


Fig. 5.2.5

If option (5) of the Set-up Utility is requested a pop-up Stop Bit's menu appears. This allows the user to choose between 1 or 2 stop bit's.

During transmission, a stop bit is sent after a character's data bits and parity bit, if there is one, to signal the end of the transmitted character. Stop bits serve to ensure that the clocks of the PC and BTS 60 systems are synchronized.

## 5.3.2.6 The Input Buffer Size.

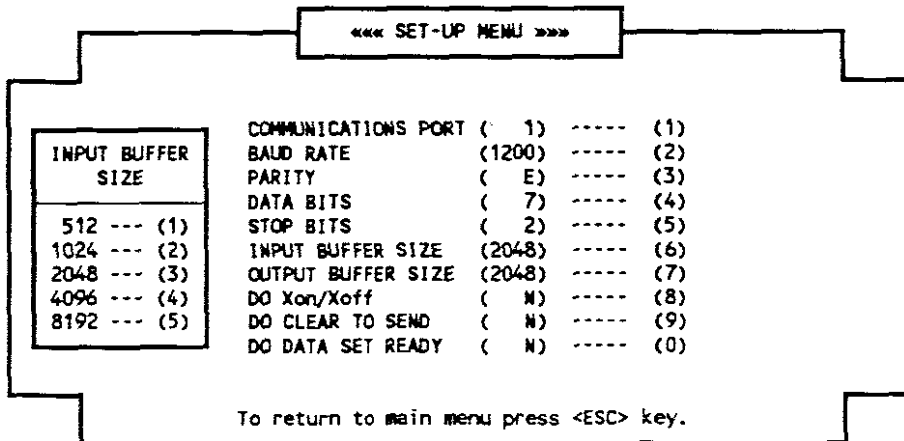


Fig. 5.2.6

If option (6) of the Set-up Utility is requested a pop-up Input Buffer Size menu appears. This menu allows the user to select the appropriate size for the input buffer.

The input buffer is a temporary storage buffer for data after it arrives over the serial port and before it is read by the communications software. If this buffer is not large enough, it could lead to the loss of incoming data. The default setting for this temporary storage buffer is set at 2048 bits.

5.3.2.7 The Output Buffer Size.

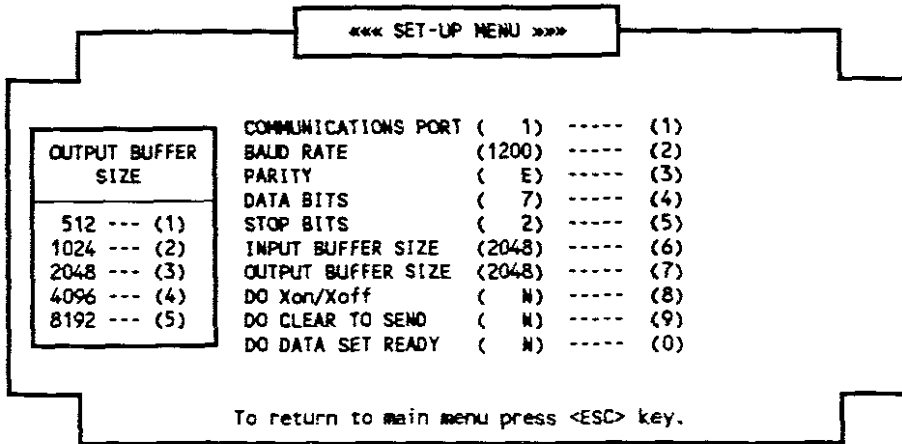


Fig. 5.2.7

If option (7) of the Set-up Utility is requested a pop-up Output Buffer Size menu appears. This menu allows the user to select the appropriate size for the output buffer.

The output buffer is a temporary storage buffer for data after it is written by the communications software and before it is sent to the serial port. If this buffer is not large enough, it could lead to the loss of output data. The default setting for this temporary storage buffer is set at 2048 bits.



## 5.3.2.8 Do Xon/Xoff.

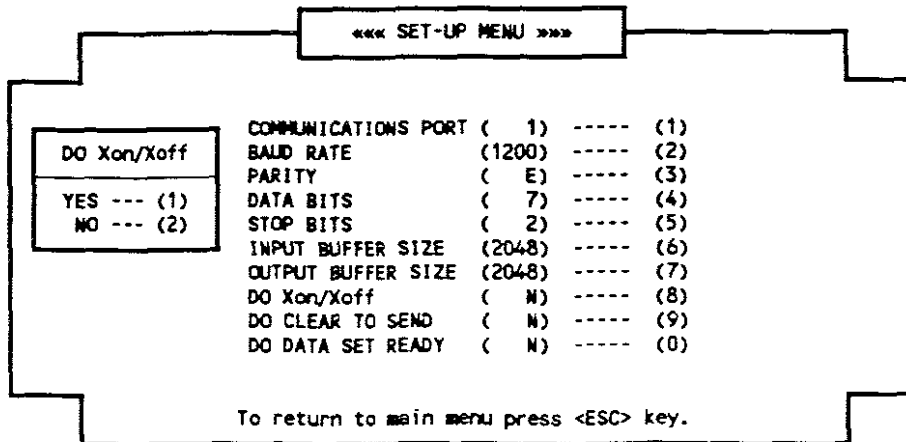


Fig. 5.2.8

If option (8) of the Set-up Utility is requested a pop-up Do Xon/Xoff menu appears. This menu allows the user to specify whether Xon/Xoff handshaking is required.

The purpose of handshaking is to ensure correct and complete data transfer. With the Xon/Xoff handshake method, the PC controls the data exchange sequence by telling the BTS 60 when it has room in its logical I/O buffer for data and when to shut off the flow.

## 5.3.2.9 Do Clear to Send.

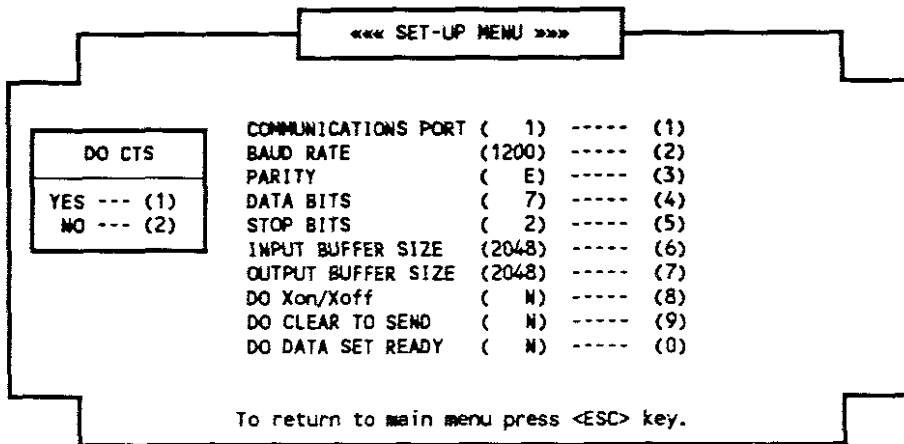


Fig. 5.2.9

If option (9) of the Set-up Utility is requested a pop-up Do Clear to Send menu appears. This menu allows the user to specify whether or not the Clear to Send signal is required.

The Clear to Send signal, if needed, would be an incoming signal to the BTS 60. This signal could be used to indicate to the BTS 60 whether the PC is ready to accept data.

## 5.3.2.10 Do Data Set Ready.

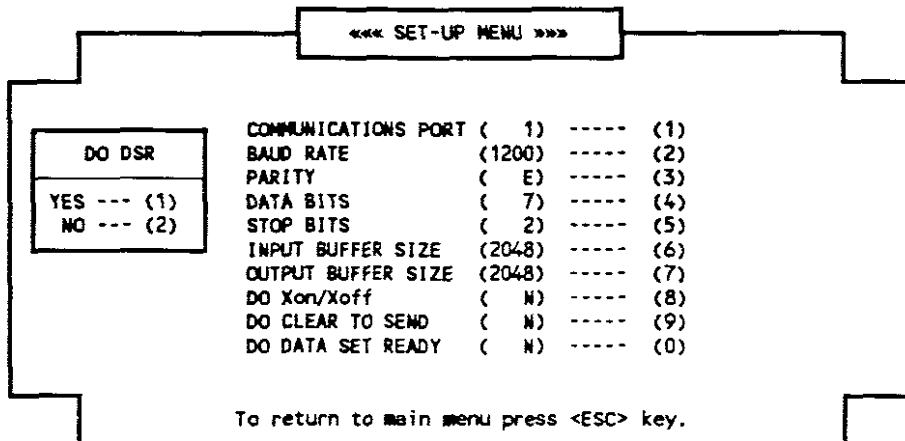


Fig. 5.2.10

If option (0) of the Set-up Utility is requested a pop-up Do Data Set Ready menu appears. This menu allows the user to specify whether or not the Data Set Ready signal is required.

The Data Set Ready signal, if needed, would be an incoming signal to the PC. This signal could be used to indicate to the PC that the BTS 60 is alive and well.

## 5.3.3 Print Reports.

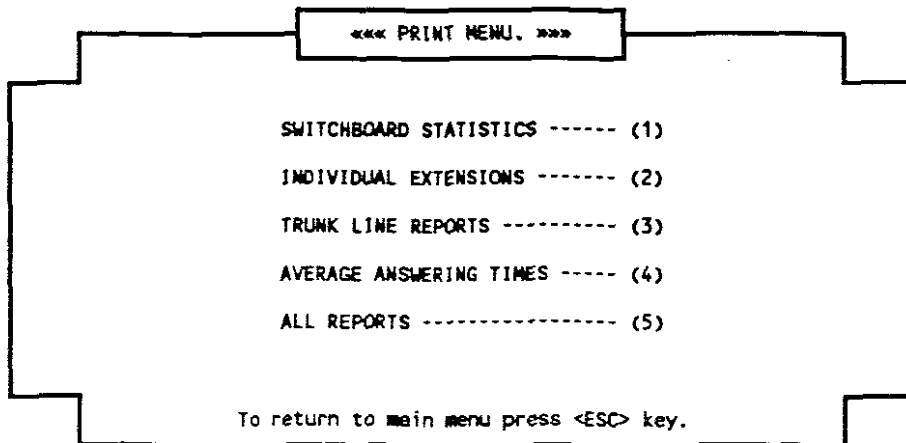


Fig. 5.3

Various detailed reports and summary reports can be produced by using the Print Menu to provide useful, easy to read, tabulated management reports. These include user defined reports such as Switchboard Statistics, Individual Extensions, Trunk Line Reports and Average Answering Times.

Typically, the information provided becomes an invaluable tool for management to reduce telephone abuse and optimise business efficiency.

## 5.3.3.1 Daily Switchboard Statistics.

DAILY SWITCHBOARD STATISTICS.				DATE: 18/04/90		
Time	Number of Ext	Amount Answered	Amount Unanswered	Average Duration	Longest Call	Shortest Call
8 - 9	17	204	2	02:20	(237) 15:28	(239) 00:05
9 - 10	17	246	96	02:17	(229) 12:48	(231) 00:01
10 - 11	17	263	10	02:09	(235) 15:27	(252) 00:02
11 - 12	18	243	10	02:19	(244) 10:51	(251) 00:02
12 - 13	18	208	5	02:20	(242) 10:33	(252) 00:03
13 - 14	17	145	52	01:59	(237) 12:37	(234) 00:02
14 - 15	17	223	112	02:13	(235) 10:54	(233) 00:01
15 - 16	16	225	94	02:46	(235) 26:54	(238) 00:02
16 - 17	12	32	12	01:40	(239) 05:57	(251) 00:07
		<u>1789</u>	<u>393</u>	<u>02:18</u>	<u>26:54</u>	<u>00:01</u>

Fig. 5.3.1

The "Switchboard Statistics" report was designed to offer general or global information regarding the switchboard as a whole. This report is aimed specifically for the use of managerial and supervisory staff. From this report it becomes possible to pinpoint congestion times which could result in possible lost business. The average duration per call answered could possibly be used as a measure of the switchboard's efficiency. It also highlights lengthy and time-wasting incoming calls as well as which extensions answered them.

## 5.3.3.2 Individual Extensions.

EXTENSION : 228		Date: 18/04/90				
Time	Amount Answered	Total Duration	Average Duration	Idle Time	Longest Call	Shortest Call
08:00 - 09:00	13	0:28:12	0:02:10	31:48	0:04:51	0:00:58
09:00 - 10:00	12	0:34:41	0:02:53	25:19	0:09:49	0:00:03
10:00 - 11:00	19	0:32:53	0:01:44	27:07	0:06:25	0:00:40
11:00 - 12:00	18	0:38:31	0:02:08	21:29	0:04:44	0:00:49
12:00 - 13:00	17	0:29:41	0:01:45	30:19	0:03:55	0:00:23
13:00 - 14:00	3	0:16:21	0:05:27	43:39	0:08:50	0:00:03
14:00 - 15:00	20	0:38:17	0:01:55	21:43	0:09:30	0:00:16
15:00 - 16:00	16	0:47:04	0:02:56	12:56	0:07:54	0:00:39
16:00 - 17:00	4	0:10:35	0:02:39	49:25	0:05:14	0:01:05
	<u>122</u>	<u>4:36:15</u>	<u>0:02:16</u>		<u>0:09:49</u>	<u>0:00:03</u>

Fig. 5.3.2

The "Individual Extensions" report was designed to present to the user a report on the hourly and daily performance of each extension. This allows the supervisory staff as well as the operators the opportunity to compare the various extensions. If utilized correctly, this could form the basis for healthy competition amongst the operators manning the switchboard. This in turn would create greater staff efficiency and productivity which will reduce customer frustration and improve the company's image.

## 5.3.3.3 Trunk Line Reports.

TRUNK LINE : 1		DATE: 18/04/90			
Time	Amount Answered	Duration Occupied	Amount Unanswered	Duration Occupied	Total Duration Occupied
08:00 - 09:00	27	0:45:59	1	0:00:13	0:46:12
09:00 - 10:00	13	0:50:02	5	0:04:59	0:55:01
10:00 - 11:00	22	0:51:14	1	0:00:47	0:52:01
11:00 - 12:00	19	0:52:28	3	0:01:27	0:53:55
12:00 - 13:00	21	0:46:03	0	0:00:00	0:46:03
13:00 - 14:00	20	0:42:32	4	0:02:37	0:45:09
14:00 - 15:00	17	0:41:53	10	0:11:06	0:52:59
15:00 - 16:00	16	0:47:35	6	0:06:53	0:54:28
16:00 - 17:00	3	0:09:18	0	0:00:00	0:09:18
	<u>158</u>	<u>6:27:04</u>	<u>30</u>	<u>0:28:02</u>	<u>6:55:06</u>

Fig. 5.3.3

The "Trunk Line" reports indicate to what extent the various trunk lines have been utilized. It offers information pertaining to the duration for which the line is occupied, due to both answered as well as unanswered calls. These figures could be of great importance when having to decide whether the number of incoming lines serving the switchboard are sufficient. This report can also be used to locate faulty exchange lines.

**5.3.3.4 Average Answering Times.**

AVERAGE DAILY ANSWERING TIMES.			DATE: 18/04/90
EXTENSION	AMOUNT ANSWERED	AVERAGE RINGING TIME	AVERAGE SPEECH TIME
228	122	0:00:27	0:02:16
229	130	0:00:21	0:02:33
230	119	0:00:19	0:02:36
231	60	0:00:20	0:02:07
232	112	0:00:29	0:02:01
233	127	0:00:30	0:01:44
234	133	0:00:31	0:01:55
235	98	0:00:21	0:03:06
236	22	0:00:17	0:01:46
237	92	0:00:21	0:03:15
238	75	0:00:32	0:02:01
239	105	0:00:22	0:02:20
242	106	0:00:15	0:01:58
243	106	0:00:18	0:02:24
244	104	0:00:28	0:02:01
251	85	0:00:16	0:02:24
252	109	0:00:17	0:02:34
253	84	0:00:25	0:02:00
Total	1789	0:00:23	0:02:18

Fig. 5.3.4

The "Average Answering Times" report was designed to indicate the efficiency of the switchboard. It offers the user information on the average ringing and speech times per call. This information is useful when trying to enhance your company's image to callers.

**5.3.3.5 All Reports.**

This option combines all the above reports into one report for possible archiving purposes.



## 5.3.3.6 Printer Error.

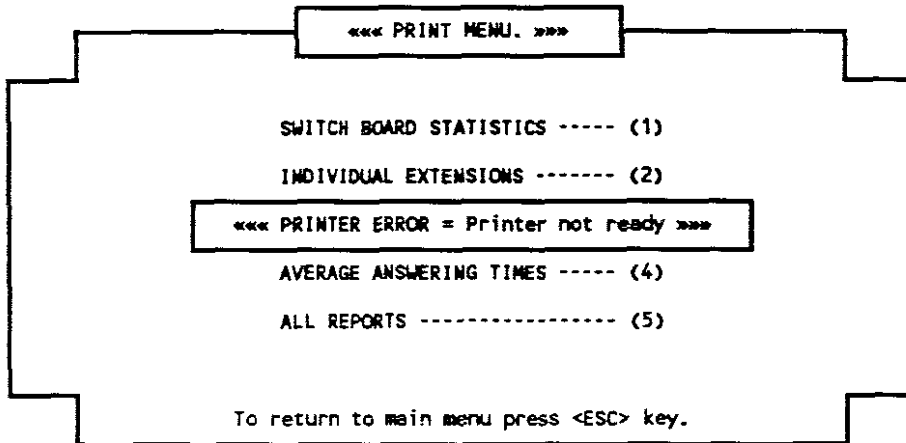


Fig. 5.3.5

When a report from the Print Menu is selected by the user to be printed it is important to establish whether the connected printer is correctly initialized.

If there is any problem in communicating with the printer it is pointless to process the relevant data for the report. Thus, if there is any problem with the printer, a "Printer Error" window opens over the Print Menu to indicate that there is a problem and the program returns to the Main Menu.

## 5.3.3.7 Processing Data for Printed Report.

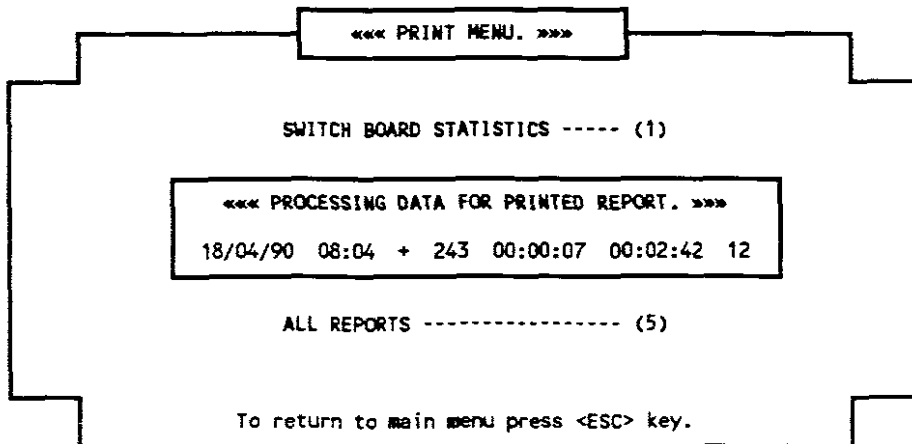


Fig. 5.3.6

On selecting any of the available reports to be printed from the Print Menu, the data in the database has to be processed.

While the data is being processed a "Processing Data for Printed Report" window opens over the Print Menu. This indicates to the user that the system is not inactive, but is in fact processing data. Once the data has been processed it is dumped to the printer and the program returns to the Main Menu.

#### 5.3.4 System Date and Time.

The system date and time is obtained from the PC's internal clock. This date and time is of the utmost importance as the program relies on this information for triggering various procedures. To display the system date and time, select the "System Date and Time" option from the Main Menu. On selecting this option the following menu appears (See Fig. 5.4.1).

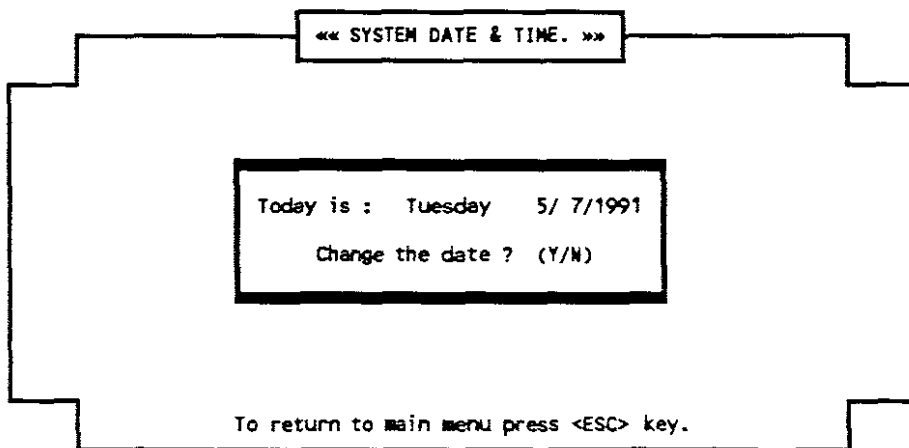
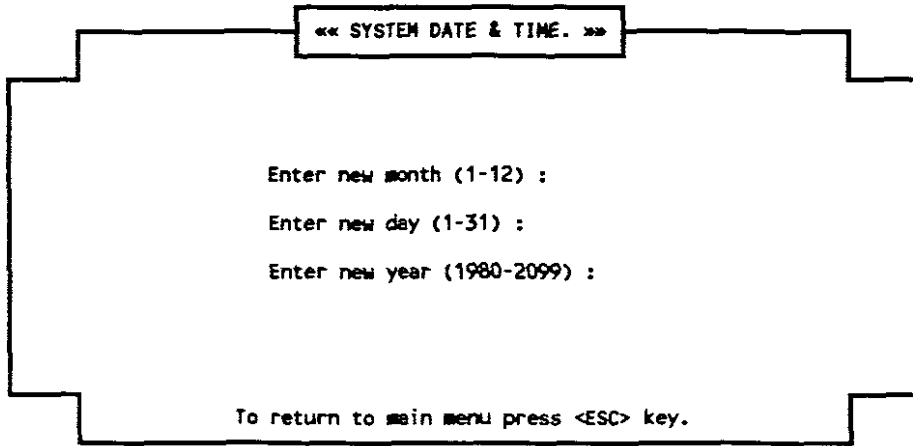


Fig. 5.4.1

This menu displays the system date and prompts the user to change the date. If the user decides to change the date the following menu will appear (See Fig. 5.4.2).



«« SYSTEM DATE & TIME. »»

Enter new month (1-12) :

Enter new day (1-31) :

Enter new year (1980-2099) :

To return to main menu press <ESC> key.

Fig. 5.4.2

This menu allows the user to change the system date from the keyboard.

The date should be entered according to the following criteria:

month -- must be 1 or 2 integers from 1 to 12.

day ---- must be 1 or 2 integers from 1 to 31.

year --- must be 4 digits from 1980 to 2099.

If an improper date is entered, the date will be ignored and the original date retained. If a valid date is entered the new date is accepted and the following menu appears (See Fig. 5.4.3).

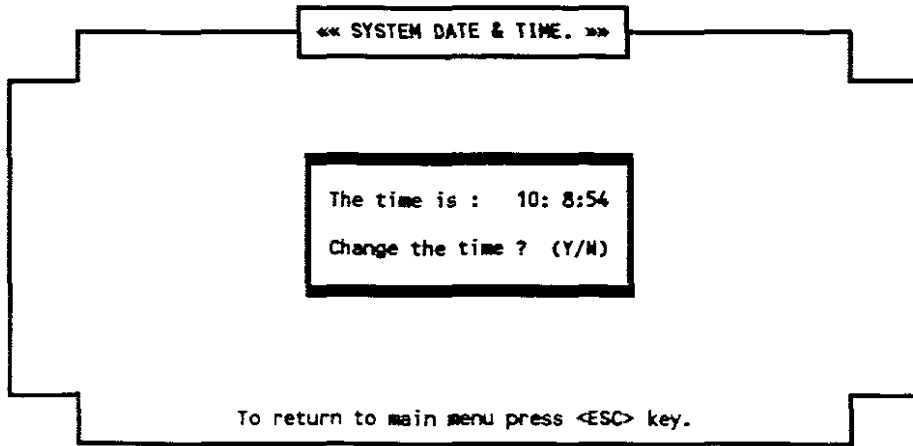


Fig. 5.4.3

This menu displays the system time and prompts the user to change the time. The time is displayed as hours, minutes and seconds in the 24-hour clock format.

If the user decides not to change the system time, the program will exit the "System Date and Time" menu and return to the Main Menu.

If the user chooses to change the time, the following menu will appear (See Fig. 5.4.4).

«« SYSTEM DATE & TIME. »»

Enter new hour (0-23) :

Enter new minute (0-59) :

Enter new second (0-59) :

To return to main menu press <ESC> key.

Fig. 5.4.4

This menu allows the user to change the system time from the keyboard.

The time should be entered according to the following criteria.

hour ---- must be 1 or 2 integers from 0 to 23.

minute -- must be 1 or 2 integers from 0 to 59.

second -- must be 1 or 2 integers from 0 to 59.

If an invalid time is entered the time will be ignored and the original time retained. If a valid time is entered the new time is accepted and the program returns to the Main Menu.

5.3.5 Office Lay-out and Extension Numbers.

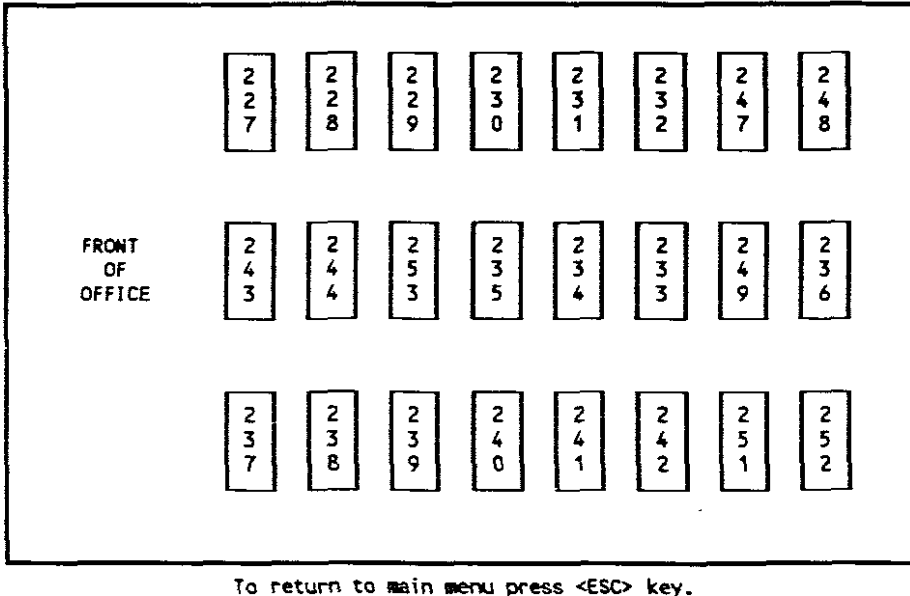


Fig. 5.5

The "Office Lay-out and Extension Numbers" screen was designed exclusively for the supervisory staff. The aim of this screen is to assist the supervisor in locating a specific extension and operator. This could be advantageous when analysing, for example, the Individual Extensions report.

#### 5.3.6 Quit.

The Quit option simply terminates the program Monitor and returns the user to the PC's operating system.

#### 5.4 Archiving.

The BTS 60 system resets its parameters at midnight and this triggers the Monitor software to take care of the necessary archiving.

The archiving is done in two ways, viz. to a disk file that is appended each day and as a hardcopy print-out of all the reports. Once the daily data has been archived to its archive disk file, the daily data captured is cleared and a disk file is prepared for the next day's data. The archiving to a disk file has been designed in such a way as to handle one years accumulated data in less than 10 Mbytes of disk space.

#### 5.5 Handshaking.

"Monitor" utilizes the BTS 60's internal buffer, of a 100 calls, by means of handshaking signals. This ensures that incoming data is not lost while the user is selecting menus, printing reports or processing data.



**5.6 Program Flowcharts and Listing.**

The program flowcharts and listings are detailed in Appendices B and C respectively.

## **6 PROBLEMS ENCOUNTERED.**

The major problem encountered throughout the duration of this project concerned the lack of computer literacy experienced amongst the Account Enquiry staff.

Due to this problem it was extremely difficult to establish exactly what information the section as a whole required. To overcome this problem one had to present them with fictitious information and ideas to prompt some kind of a reaction from them. This was done to great effect.

During the feasibility study and system analysis stages of this project great difficulty was found in acquiring general information regarding telephone management information systems. This was due to the lack of literature available on the subject and the reluctance of private companies to assist in acquiring such information. Most of the private companies that were approached by means of pre-arranged interviews or telephonic discussions were extremely willing to assist until they established that the proposed project was for the Department of Posts and Telecommunications.

## PROBLEMS ENCOUNTERED

On completion of the project a problem was experienced in trying to persuade the users of the management information system, be they managerial, supervisory or operational users, to see this system as a management tool rather than as a type of "watch dog".

To achieve this, one had to promote a new and more flexible management structure that would make use of management tools to provide a more efficient and productive Account Enquiry section. This was done to a large extent by constantly marketing the system at every available opportunity as an excellent management tool without which proper decision making is impossible.

## **7. FUTURE ENHANCEMENTS.**

Software development by nature is an on-going process that is susceptible to future enhancements and the software written for this project is no different. The following possible enhancements could be taken care of in future versions of the system software.

- Dynamically changing the software.
- Monthly and quarterly generated reporting.
- Graphical representation of information.
- Dynamical report generation.

### **7.1 Dynamically Changing the Software.**

The system software can be expanded to facilitate the dynamic re-structuring necessary for this program to be able to interface with future BTS systems, such as the BTS 128, without having to access the source code of the program.

### **7.2 Monthly and Quarterly Generated Reporting.**

At present the software only caters for daily reporting that indicates the hourly performance of the switchboard, operators, trunk lines, etc.

## FUTURE ENHANCEMENTS

Given the necessary time and data, the software could be updated to produce monthly or even quarterly reports to assist the Account Enquiry section even further in their quest to improve the service they are rendering to the telephone-using public.

### 7.3 Graphical representation of information.

Sometimes, instead of seeing information in a tabular format, one may want to see information displayed in ways that will make the results easier to interpret. Graphs make it easy to visualize information for the following reasons.

- One can see trends.
- One can make comparisons.
- One can analyse relationships.

Once the telephone management information system designed in this project has taken off and is being utilized as a management tool, the software could be revised to incorporate graphs as well.

7.4 Dynamic report generation.

All the reports that are created with the aid of the management information system are created by default. A future enhancement to the system would be to allow the users to dynamically change these default settings to cater for each users personal requirements.

## **8 CONCLUSION.**

The data logging and monitoring system that was designed, developed and built for this project, significantly surpasses the original requirements as laid down by the Department of Posts and Communications. This, to a large extent, is due to the enormous amount of research that went into the designing and development of management information systems in general.

The success of this particular information system is influenced by a number of factors. The major factor being that the software is designed to be extremely user friendly and user tolerant, without detracting from its flexibility.

The time and effort spent on designing and developing this project has lead to the implementation of an exceptionally effective and practical management tool. The need for such a management information tool in the Department of Posts and Telecommunications does not end at the Account Enquiry section as various other sections have the need for management information.

## CONCLUSION

The experience and knowledge gained by undertaking this project has greatly enriched each and every individual that took part in it. This experience and knowledge gained will contribute significantly to the success of future projects.

There is no doubt that this project will play an influential part in a more efficient and effective service being presented by the Account Enquiry section and the Department of Posts and Telecommunications as a whole.



## **9 BIBLIOGRAPHY.**

Barnes, Ralph M. 1980. **Motion and Time Study Design and Measurement of Work Seventh Edition.** John Wiley & Sons, Inc.

Borland International. **Turbo Pascal 4.0 IBM Version.**  
Borland International, Inc.

Borland International. **Turbo Pascal Reference Manual.**  
Borland International, Inc.

Gibson, Ivancevich, Donnelly. 1985. **Organizations Behavior - Structure - Processes Fifth Edition.**  
Business Publications, Inc.

Hergert, Douglas. **Mastering Turbo Pascal 5.** Sybex.

Hunt, Gary T. 1980. **Communication Skills in the Organization.** Prentice-Hall, Inc.

Intersil. 1983. **Hot Ideas in CMOS.** Intersil, Inc.

National Semiconductor Corporation. 1984. **CMOS Databook.** Santa Clara.

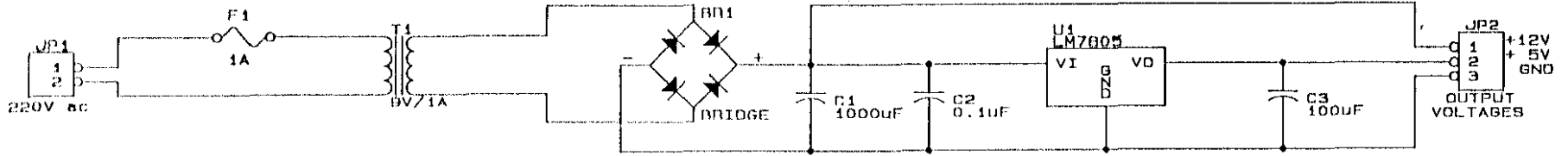
## BIBLIOGRAPHY

National Semiconductor Corporation. 1988. **CMOS Logic Databook**. Santa Clara.

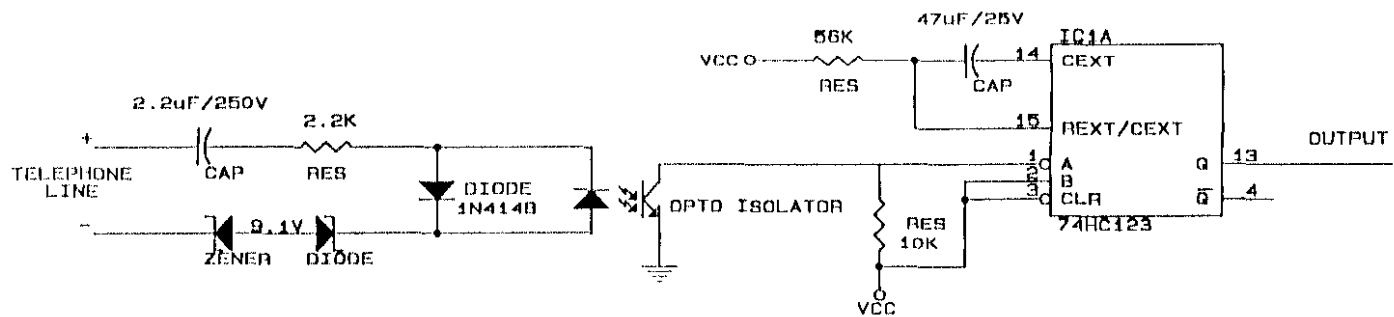
O'Brien. 1985. **Computers in Business Management Fourth Edition**. Irwin, Inc.

**APPENDIX A**

Hardware Version 1.0  
Schematic Diagram.

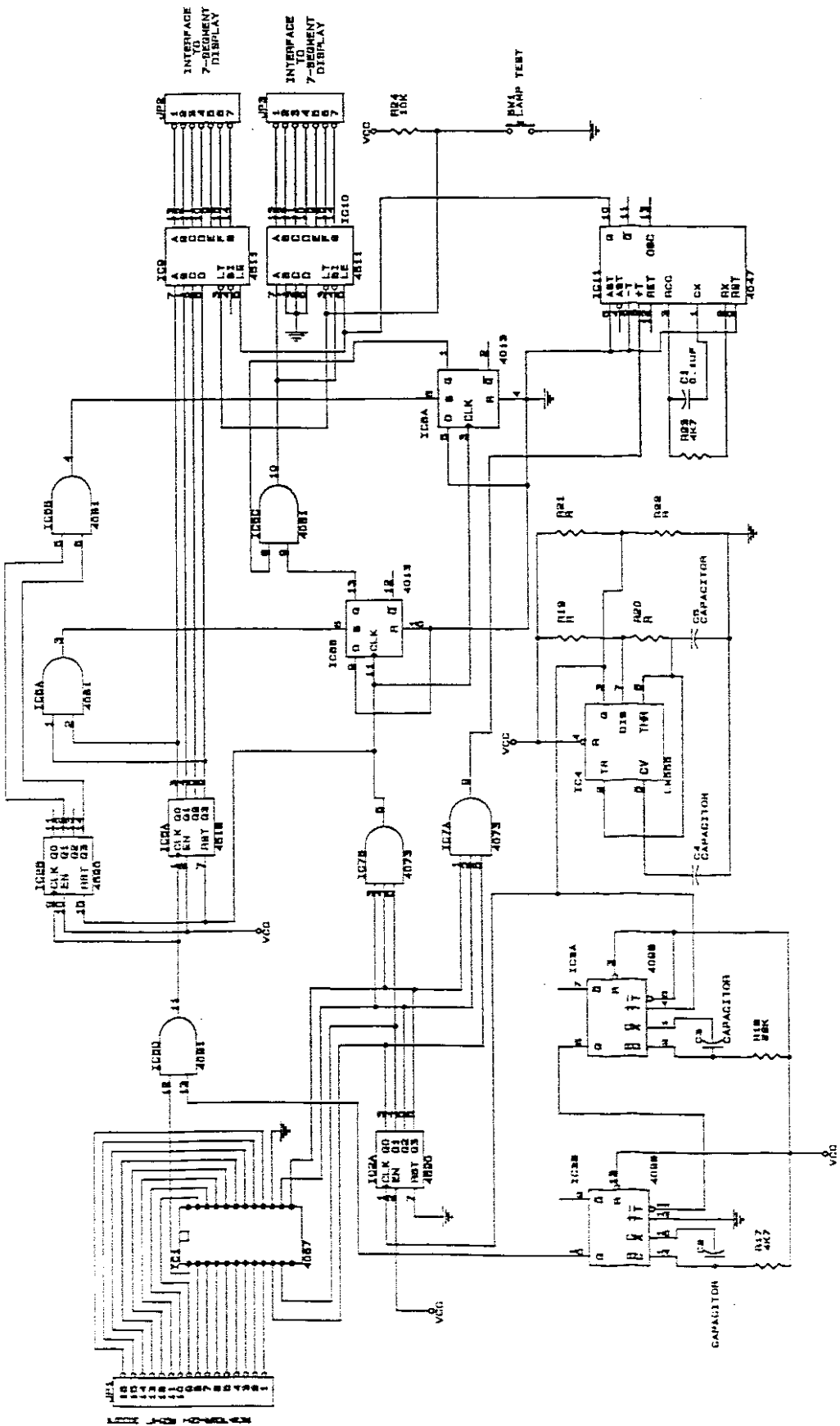


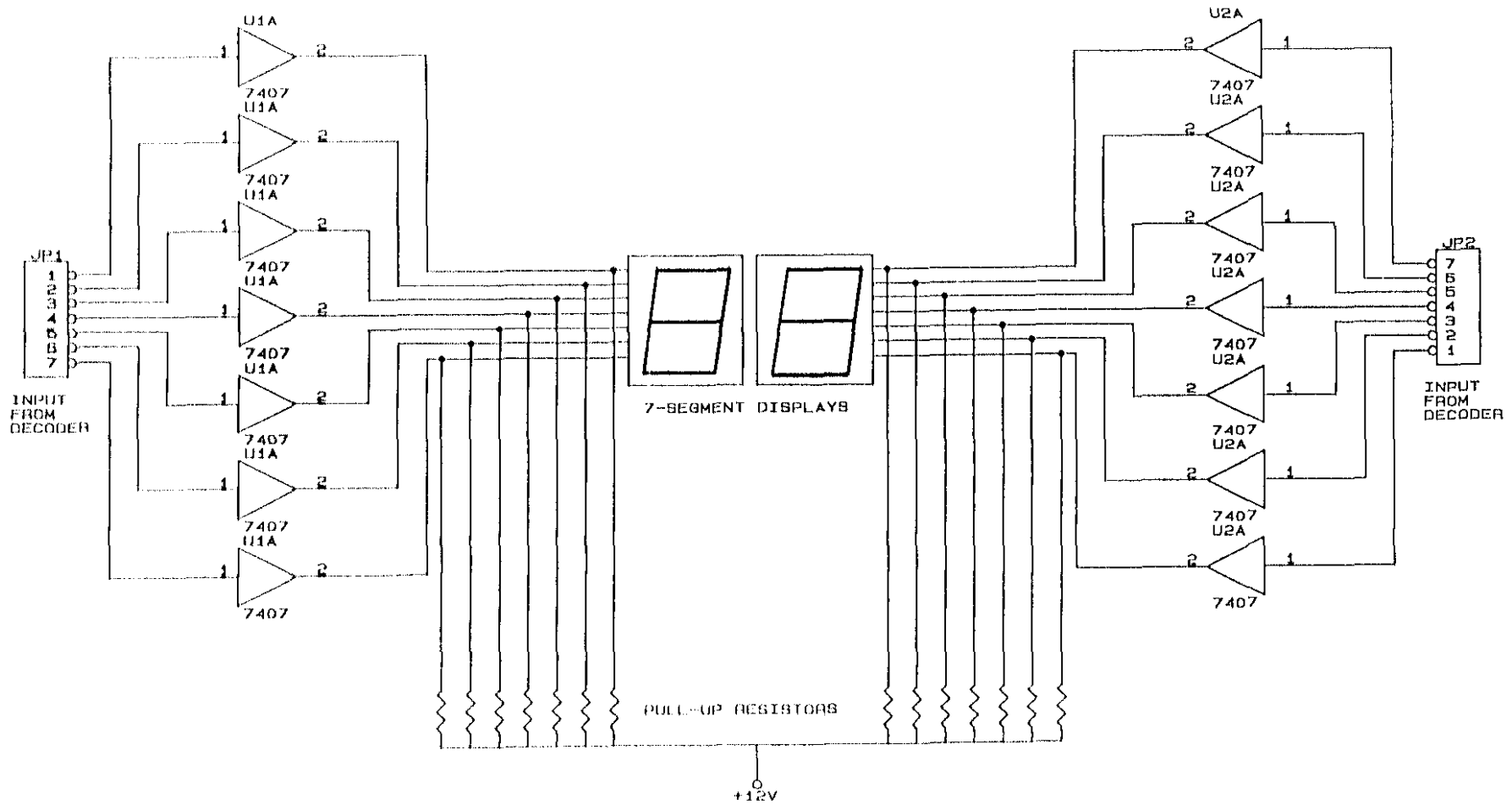
POWER SUPPLY		
Designed by: A. M. Z. MOLNAR		
Size	Document Number	REV
A		
Date:	November 20, 1991	Sheet 1 of 4



RINGING DETECTION CIRCUIT		
Designed by: A. M. Z. MOLNAR		
Size	Document Number	REV
A		
Date:	November 20, 1991	Sheet 2 of 4

MULTIPLEXING, COUNTING, DECODING AND DRIVER  
CIRCUIT FOR 7-BEIGHT DISPLAYS  
Designed by: A. M. Z. MOHAMMAD  
Size: Document Number: \_\_\_\_\_  
REV: \_\_\_\_\_  
Date: November, EO, 1991 Sheet: 3 of 4





7-SEGMENT DISPLAY CIRCUIT		
Designed by: A. M. Z. MOLNAR		
Size	Document Number	REV
A		
Date:	November 20, 1991	Sheet 4 of 4

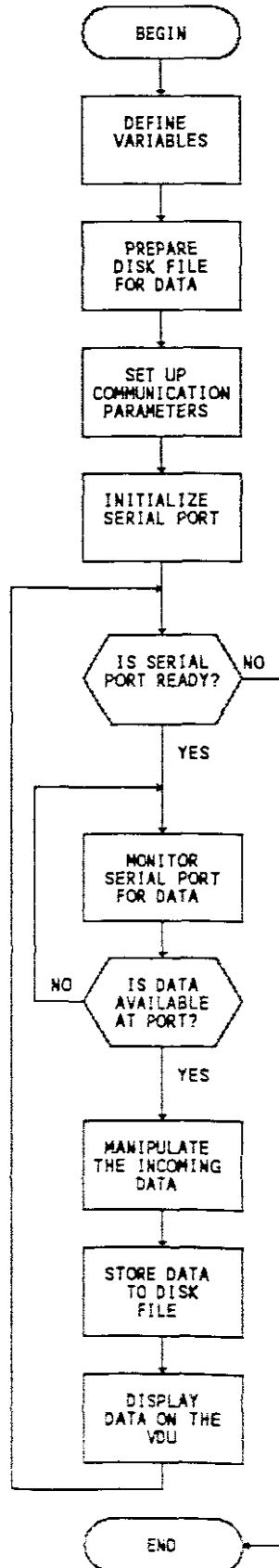
**APPENDIX B**

Software Version 1.0

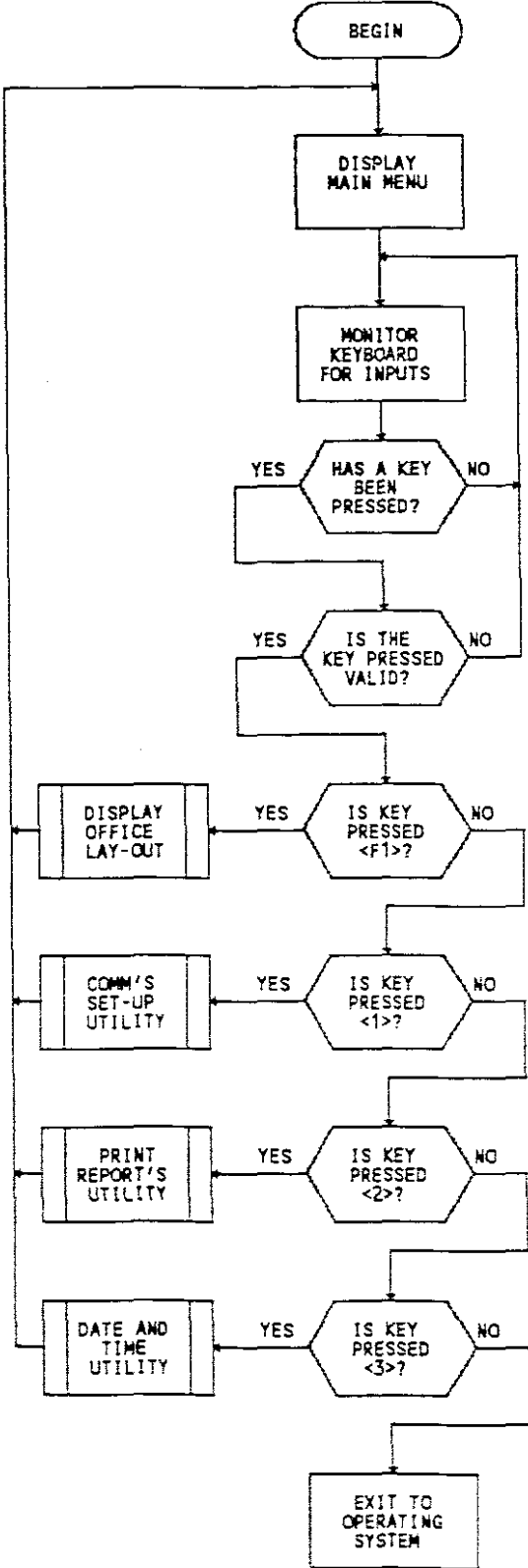
Program Flowcharts.



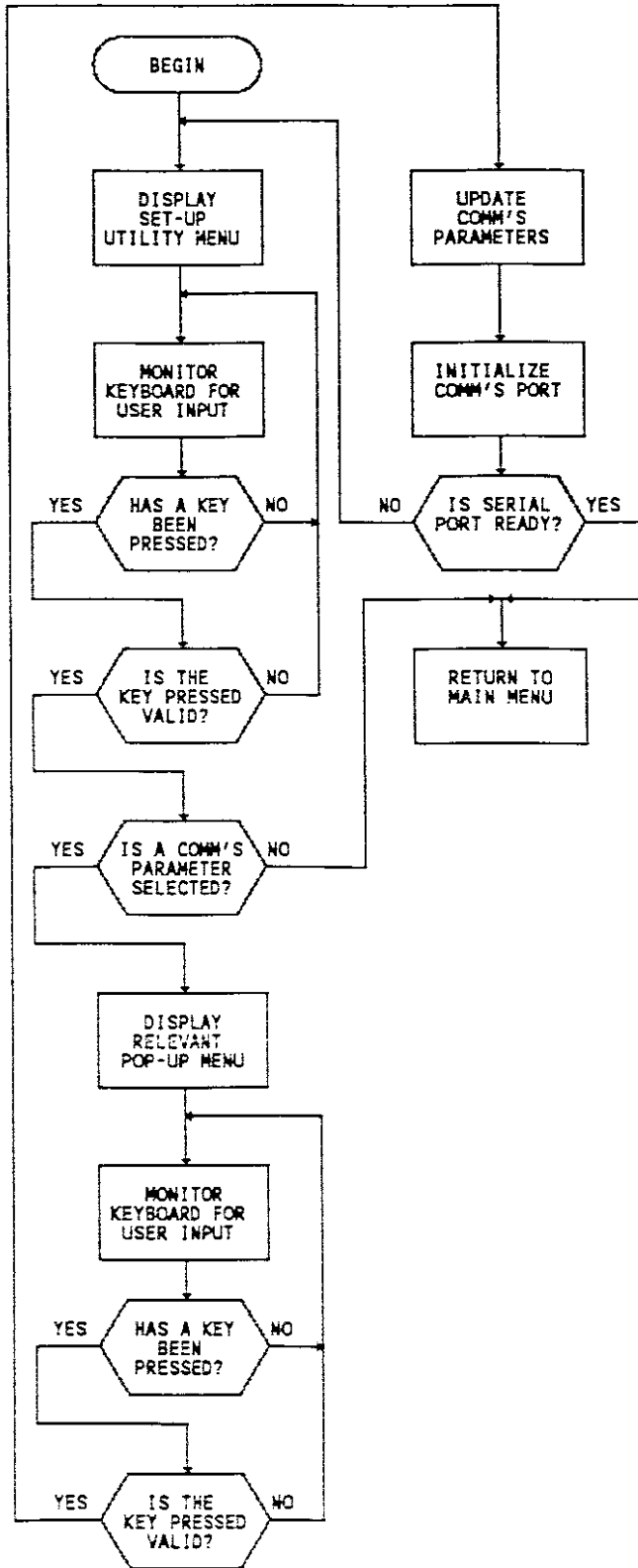
## CAPTURING OF INCOMING DATA FROM THE BTS.



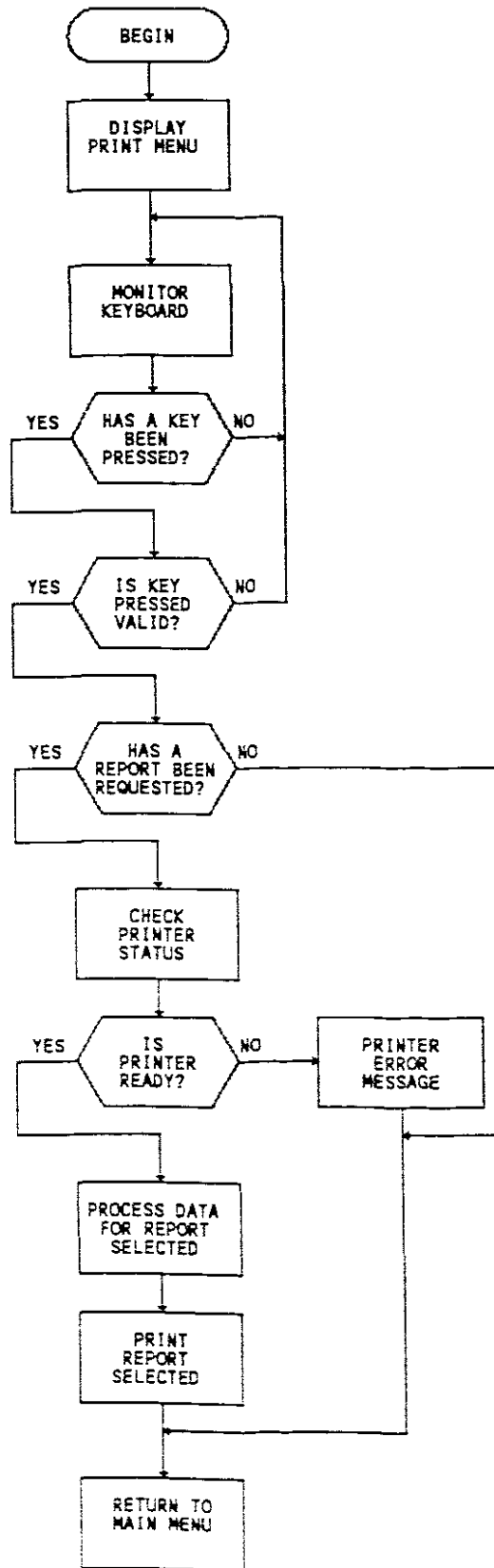
MAIN MENU INTERFACE TO THE USER.

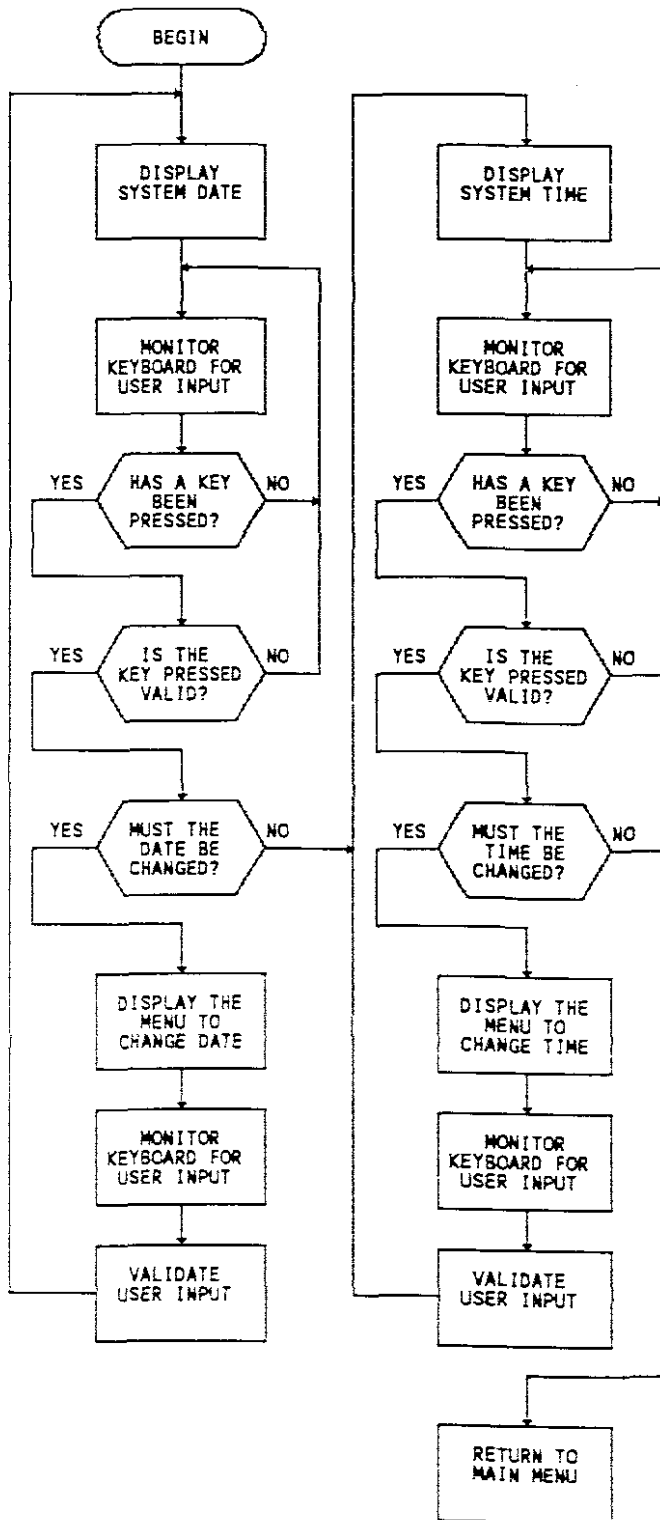


SET-UP UTILITY FOR COMMUNICATION PORT.

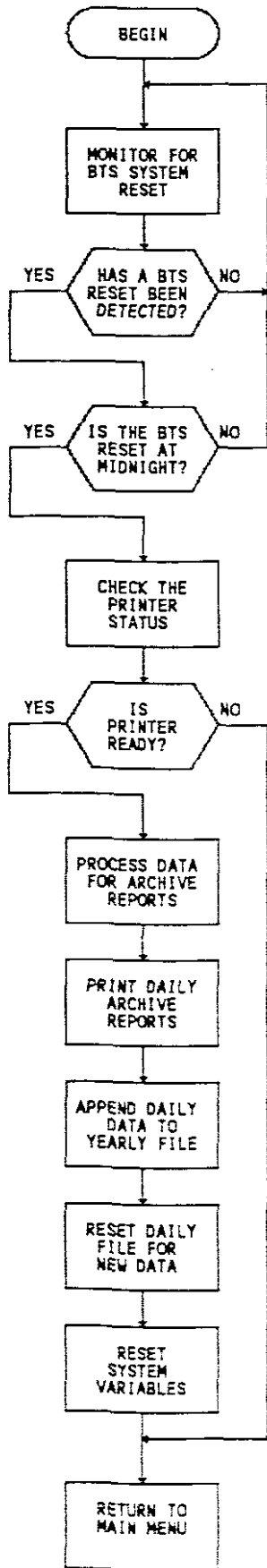


PRINT REPORTS UTILITY.





BTS 60 SYSTEM RESET AT MIDNIGHT.



**APPENDIX C**

Software Version 1.0  
Program Listing.

```

(*-----*)
(* THIS PROGRAM IS DESIGNED TO CAPTURE DATA FROM THE BTS TELEPHONE SYSTEM *)
(* AND MANIPULATE THIS DATA INTO MEANINGFUL INFORMATION. *)
(* *)
(* Written by: A. M. Z. Molnar Version 1.0 *)
(* *)
(* Dated: November, 1991 *)
(* *)
(*-----*)

```

```

program MonitorBTS;

(*$! GLOBTYPE.GLO *)
(*$! ASCII.GLO *)

type str20 = string[20];
   str80 = string[80];
   str8 = string[8];
   str10 = string[10];
   BTSData = record
       Date : string[8];
       Time : string[6];
       Ext : string[3];
       RDur,
       CDur : string[8];
       Ln : integer;
       Rec : char;
   end;

       AExt = array[220..270,7..16,1..5] of real;
       ALn = array[1..30,7..16,1..3] of real;
       AUna = array[1..30,7..16,1..2] of real;
   Arch = record
       Ext : AExt;
       Ln : ALn;
       Una : AUna;
       Dat : str8;
   end;

const Light1 : array[227..236] of integer = (6,7,8,9,10,11,12,13,14,15);
   Light2 : array[247..249] of integer = (17,18,19);
   Light3 : array[237..244] of integer = (8,9,10,11,12,13,14,15);
   Light4 : array[251..253] of integer = (17,18,19);
   Light5 : array[245..245] of integer = (16);
   Light6 : array[246..246] of integer = (16);
   Light7 : array[225..226] of integer = (6,7);
   Days : array[0..6] of string[9] = ('Sunday','Monday','Tuesday',
                                     'Wednesday','Thursday',
                                     'Friday','Saturday');

var
   Fil : file of BtsData;
   Data : btsData;
   Archfil : file of Arch;
   ArchData : Arch;
   F : Text;
   Cf : char;
   Cfstr : str80;
   FString : string[2];
   SString : string[2];
   TotRDur,Z : real;
   Baud_Rate : INTEGER;
   Com_Port : INTEGER;
   Parity : CHAR;
   Data_Bits : INTEGER;
   Stop_Bits : INTEGER;
   Point : array[1..50] of integer;
   InfoExt : AExt;
   InfoLn : ALn;
   InfoUna : AUna;
   CountExt : array[7..16] of real;
   Flag : array[1..255] of integer;

```



## APPENDIX C

```

R,S,T      : integer;
Exten,Tim  : integer;
Datum      : string[8];
Err,I,J    : integer;
X,Y        : integer;
Stop       : boolean;
Ans,Unans  : integer;
Raat,Woc   : integer;
CP,BR,SB,DB : integer;
IBS,OBS    : integer;
PR,XX,HW   : char;
CTS,DSR    : char;
YesNo,Blank : boolean;
Hour,Min,Sec,Month,Day,Year,WeekDay : integer;
ChangeTime,ChangeDate : char;
InBufSize  : INTEGER;
OutBufSize : INTEGER;
Do_XonXoff : CHAR;
Do_HardWired: CHAR;
Do_CTS     : CHAR;
Do_DSR     : CHAR;

(*$I PIBASYN1.GLO *)
(*$I PIBASYN1.MOD *)
(*$I PIBASYN2.MOD *)
(*$I PIBASYN3.MOD *)
(*$I MENU.MOD *)
(*$I PRINT.MOD *)
(*$I GETTIME.PAS *)
(*$I SETTIME.PAS *)
(*$I GETDATE.PAS *)
(*$I SETDATE.PAS *)

(*-----*)
function Exist : boolean;

(* This function is used to establish the existence of any file. *)

begin (* Exist *)
  ($I-)
  reset(ArchFil);          (* Disk file is prepared for processing. *)
  ($I+)
  Exist := IOResult = 0;   (* If file exist, the boolean variable *)
                          (* will be true. *)
end; (* Exist *)

(*-----*)

procedure Archive;

(* This procedure is used to print the necessary daily reports as well as *)
(* take care of the archiving facility. *)

begin (* Archive *)
  Async_Close(true);
  assign(ArchFil,'C:\TP\DATA\YEAR.DTA');
                          (* Assign the file path and name to the *)
                          (* variable name. *)
  if Exist                (* Test to see if file exists. *)
  then
    reset(ArchFil)
  else
    rewrite(ArchFil);
  GetPrintInfo;
  TestPrinter(YesNo);
  if YesNo
  then
    begin
      write(lst,#27#67#33);
      ManRep;
      ExtRep;
    end;
end;

```

```

LnRep;
end;
ArchData.Ext := InfoExt;      (*          *)
ArchData.Ln := InfoLn;      (*      Set archive variables.  *)
ArchData.Una := InfoUna;    (*          *)
ArchData.Dat := Data.Date;  (*          *)
seek(ArchFil,filesize(ArchFil));
(* File pointer is set to end of file. *)
write(ArchFil,ArchData);    (* Write data to data file. *)
rewrite(Fil);              (* Disk file is prepared for processing. *)
close(ArchFil);           (* The disk file is closed and the disk *)
(* directory is updated. *)
if (not Async_Open(Com_Port,Baud_Rate,Parity,Data_Bits,Stop_Bits))
(* Try opening the serial port. *)
then
writel('Cannot open serial port.')
(* Serial port can't communicate due to *)
(* incorrect parameters. *)
else
writel('Serial port opened. ');
(* Serial port is ready to communicate. *)
Datum := Data.Date;      (*          *)
Ans := 0;               (*      Reset program variables. *)
Unans := 0;            (*          *)
SetBottomLine;
end; (* Archive *)

(*-----*)
procedure LightOn;
(* This procedure is used to indicate which extensions are busy. *)
var M,N : integer;
begin (* LightOn *)
window(1,1,80,25);      (* Set size of required work area to the *)
(* whole screen. *)
ReverseVideo(true);
val(Data.Ext,M,Err);    (* Make the variable M equal to the *)
(* integer value of Data.Ext *)
(* (Extension number). *)
case M of
227..236 : begin
N := Light1[M]; (* Determine position of extension *)
(* to be highlighted. *)
gotoXY(3,N);
end;
247..249 : begin
N := Light2[M]; (* Determine position of extension *)
(* to be highlighted. *)
gotoXY(3,N);
end;
237..244 : begin
N := Light3[M]; (* Determine position of extension *)
(* to be highlighted. *)
gotoXY(76,N);
end;
251..253 : begin
N := Light4[M]; (* Determine position of extension *)
(* to be highlighted. *)
gotoXY(76,N);
end;
245..245 : begin
N := Light5[M]; (* Determine position of extension *)
(* to be highlighted. *)
gotoXY(3,N);
end;
246..246 : begin
N := Light6[M]; (* Determine position of extension *)
(* to be highlighted. *)
gotoXY(76,N);
end;
end;
end;

```

APPENDIX C

```

        end;
225..226 : begin
        N := Light7[M]; (* Determine position of extension *)
                (* to be highlighted. *)
        gotoXY(76,N);
        end;
end; (* case *)
write(M);
ReverseVideo(false);
window(11,16,70,21); (* Set the size of the required work *)
                (* area. *)
end; (* LightOn *)

(*-----*)

procedure LightOff;

(* This procedure is used to indicate which extensions are not busy. *)

var M,N : integer;

begin (* LightOff *)
    window(1,1,80,25); (* Set size of required work area to the *)
                (* whole screen. *)
    ReverseVideo(false);
    val(Data.Ext,M,Err); (* Make the variable M equal to the *)
                (* integer value of Data.Ext *)
                (* (Extension number). *)
    case M of
        227..236 : begin
            N := Light1[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(3,N);
            end;
        247..249 : begin
            N := Light2[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(3,N);
            end;
        237..244 : begin
            N := Light3[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(76,N);
            end;
        251..253 : begin
            N := Light4[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(76,N);
            end;
        245 : begin
            N := Light5[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(3,N);
            end;
        246 : begin
            N := Light6[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(76,N);
            end;
        225..226 : begin
            N := Light7[M]; (* Determine position of extension *)
                    (* to be highlighted. *)
            gotoXY(76,N);
            end;
    end; (* case *)
    write(M);
    window(11,16,70,21); (* Set the size of the required work *)
                (* area. *)
end; (* LightOff *)

(*-----*)

```

```

procedure Get_Comm_Params;

(* This procedure is used to obtain all the relevant parameters necessary *)
(* for communications. *)

var YesNo : CHAR;

begin (* Get_Comm_Params *)
  Com_Port := CP;          (* Which port, e.g., 1 for COM1: *)
  Baud_Rate := BR;        (* Baud rate for connection, e.g., 1200 *)
  Parity := PR;          (* Parity, e.g., E for even parity *)
  Data_Bits := DB;       (* How many bits per character, e.g., 8 *)
  Stop_Bits := SB;       (* How many stop bits -- nearly always 1 *)
  InBufSize := IBS;      (* Size of input buffer. *)
  OutBufSize := OBS;     (* Size of output buffer. *)
  Do_XonXoff := XX;      (* 'Y' to do XON/XOFF flow control. *)
  Do_CTS := CTS;         (* 'Y' to do CTS checking. *)
  Do_DSR := DSR;         (* 'Y' to do DSR checking. *)
  Do_HardWired := HW;    (* 'Y' to do HardWired flow control. *)
end (* Get_Comm_Params *);

(*-----*)

function Initialize_Communications : boolean;

(* This function is used to open the serial port for communication. *)

begin (* Initialize_Communications *)
  Async_Do_CTS := ( UpCase( Do_CTS ) = 'Y' );
                (* Set CTS checking. *)
  Async_Do_DSR := ( UpCase( Do_DSR ) = 'Y' );
                (* Set DSR checking. *)
  Async_Do_XonXoff := ( UpCase( Do_XonXoff ) = 'Y' );
                (* Set XON/XOFF to user request. *)
  Async_Hard_Wired_On := ( UpCase( Do_HardWired ) = 'Y' );
                (* Set hard-wired as user requests. *)
  Async_Break_Length := 500;
                (* Set half-second break duration. *)
  Async_Init( InBufSize, OutBufSize, 0, 0, 0);
                (* Let XON/XOFF break points default. *)
  if (not Async_Open(Com_Port,Baud_Rate,Parity,Data_Bits,Stop_Bits))
                (* Try opening the serial port. *)
  then
    begin
      writeln('Cannot open serial port. ');
      Initialize_Communications := false;
                (* Serial port can't communicate due to *)
                (* incorrect parameters. *)
    end
  else
    begin
      writeln('Serial port opened. ');
      Initialize_Communications := true;
                (* Serial port is ready to communicate. *)
    end
  end;
end (* Initialize_Communications *);

(*-----*)

procedure GetData;

(* This procedure is used to manipulate the data received at the serial port *)
(* and to re-organize it to a more suitable format. *)

begin (* GetData *)
  if (pos('CALL LOGGING REPORT',Cfstr) <> 0)
  then
    begin
      begin
        if Hour > 23
          then
            Archive;
        if Hour < 1
          then
            Archive;
      end
    end
  end;
end (* GetData *);

```

```

    then
      Archive;
    end;
  end;
  if (pos('Page',Cfstr) <> 0) then Datum := copy(Cfstr,1,8);
    (* Check for the word 'Page', if found *)
    (* then the first eight characters of *)
    (* that string will represent the date. *)
  if (pos(':',Cfstr) <> 0) then
    (* Check each string for a ':', if found *)
    (* then that string contains information *)
    (* regarding a telephone enquiry. *)
    window(1,1,80,25); (* Set size of required work area to the *)
    (* whole screen. *)
    GetTime(Hour,Min,Sec); (* Check system time. *)
    ReverseVideo(true);
    gotoXY(70,1);
    write(Hour:2,Min:2,Sec:2);(* Update the time at the top of the *)
    (* screen. *)
    ReverseVideo(false);
    window(11,16,70,21); (* Set the size of the required work *)
    (* area. *)
  begin
    window(1,1,80,25); (* Set size of required work area to the *)
    (* whole screen. *)
    gotoXY(20,24);
    write(Datum); (* Update the date at the bottom of the *)
    (* screen. *)
    window(11,16,70,21); (* Set the size of the required work *)
    (* area. *)
    Data.Rec := copy(Cfstr,1,1); (* Type of record. *)
    Data.Ext := copy(Cfstr,2,3); (* Number of extension answering the *)
    (* call or unanswered call. *)
    val(copy(Cfstr,51,2),Data.Ln,Err);
    (* Number of line on which call was made. *)
    if (Data.Rec <> '+')
      (* Check the variable Data.Rec for a '+' *)
      (* , if found that string contains *)
      (* information about a call being *)
      (* answered. *)
    then
      begin
        if Data.Rec = '+'
          then
            begin
              if Data.Ext = 'Una'
                (* Check the variable Data.Ext for 'Una' *)
                (* , if found that string contains *)
                (* information about a unanswered call. *)
              then
                begin
                  Data.Date := Datum;
                  (* The date on which the call is made. *)
                  Data.Time := copy(Cfstr,33,5);
                  (* The time at which the call is made. *)
                  Data.RDur := copy(Cfstr,40,8);
                  (* The ringing time of the call. *)
                  Data.CDur := '00:00:00';
                  (* Reset the speech time. *)
                  seek(Fil,filesize(Fil));
                  (* File pointer is set to end of file. *)
                  write(Fil,Data);
                  (* Write data to data file. *)
                  Unans := Unans + 1;
                  (* Number of unanswered calls. *)
                  window(1,1,80,25);
                  (* Set size of required work area to the *)
                  (* whole screen. *)
                  gotoXY(66,24);
                  write(Unans:3);
                  (* Update the number of unanswered calls *)
                  (* at the bottom of the screen. *)
                  window(11,16,70,21);
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;

```

## APPENDIX C

```

                                (* Set the size of the required work *)
                                (* area. *)
                                *)
end
else
  if (Point[Data.Ln] <> 0)
    (* Check if this call corresponds to a *)
    (* previously answered call. *)
    *)
  then
    begin
      seek(Fil,Point[Data.Ln]);
      (* File pointer is set to the position *)
      (* where the corresponding answered *)
      (* data of this call is held. *)
      *)
      read(Fil,Data);
      (* Read data from data file. *)
      *)
      Data.Date := Datum;
      (* The date on which the call is made. *)
      *)
      Data.Ext := copy(Cfstr,2,3);
      (* Number of extension answering the *)
      (* call. *)
      *)
      Data.CDur := copy(Cfstr,40,8);
      (* The speech time of the call. *)
      *)
      seek(Fil,Point[Data.Ln]);
      (* File pointer is set to the position *)
      (* where the corresponding answered *)
      (* data of this call is held. *)
      *)
      write(Fil,Data);
      (* Write data to data file. *)
      *)
      seek(Fil,filesize(Fil));
      (* File pointer is set to end of file. *)
      *)

      LightOff;
    end;
  end;
end
else
  begin
    if (Data.Rec = '*') (* Check the variable Data.Rec for '*', *)
      (* if found that string contains *)
      (* information about a transferred call. *)
      *)
    then
      begin
        seek(Fil,Point[Data.Ln]);
        (* File pointer is set to the position *)
        (* where the corresponding answered *)
        (* data of this call is held. *)
        *)
        read(Fil,Data); (* Read data from data file. *)
        *)
        Data.Date := Datum;
        (* The date on which the call is made. *)
        *)
        Data.CDur := copy(Cfstr,40,8);
        (* The speech time of the call. *)
        *)
        Data.Rec := '*';
        (* Indicate which extension transferred *)
        (* the call. *)
        *)
        seek(Fil,Point[Data.Ln]);
        (* File pointer is set to the position *)
        (* where the corresponding answered *)
        (* data of this call is held. *)
        *)
        write(Fil,Data);
        (* Write data to data file. *)
        *)
        Data.Rec := '-';
        (* Indicate which extension received the *)
        (* transferred call. *)
        *)
        Data.Ext := copy(Cfstr,2,3);
        (* Number of extension answering the *)
        (* call. *)
        *)
        Data.RDur := copy(Cfstr,40,8);
        (* The ringing time of the call. *)
        *)
        seek(Fil,filesize(Fil));
        (* File pointer is set to end of file. *)
        *)
        Point[Data.Ln] := filepos(Fil);
        (* Pointer that indicates where the *)
        *)
      end;
    end;
  end;
end

```

APPENDIX C

```

                                (* answered part of the whole record *)
                                (* is stored within the data file. *)
write(Fil,Data);
                                (* Write data to data file. *)
seek(Fil,filesize(Fil));
                                (* File pointer is set to end of file. *)
end
else
begin
Data.Time := copy(Cfstr,33,5);
                                (* The time at which the call is made. *)
Data.RDur := copy(Cfstr,40,8);
                                (* The ringing time of the call. *)
Data.CDur := '00:00:00';
                                (* Reset the speech time. *)
Point[Data.Ln] := filepos(Fil);
                                (* Pointer that indicates where the *)
                                (* answered part of the whole record *)
                                (* is stored within the data file. *)
write(Fil,Data);
                                (* Write data to data file. *)

Ans := Ans + 1;
                                (* Number of answered calls. *)
window(1,1,80,25);
                                (* Set size of required work area to the *)
                                (* whole screen. *)
gotoXY(44,24);
write(Ans:4);
                                (* Update the number of unanswered calls *)
                                (* at the bottom of the screen. *)
window(11,16,70,21);
                                (* Set the size of the required work *)
                                (* area. *)
LightOn;
end;
end;
end; (* GetData *)

(*-----*)
procedure Finish_Communications;

(* This procedure is used to terminate communication and reset the system. *)

begin (* Finish_Communications *)
Async_Close(true);
                                (* Close port and drop DTR. *)
Async_Release_Buffers;
                                (* Release space allocated for buffers. *)
end (* Finish_Communications *);

(*-----*)

procedure RightJustifyNumbers2( X,Y,Z : integer; var Value : integer);

(* This procedure is used to read numbers in from the keyboard and to *)
(* display them on the screen right justified. *)

Label Out,Out1;

Var St1 : char;
St2 : string[2];
Pass : boolean;
Code : integer;

begin (* RightJustifyNumbers2 *)
gotoXY(X,Y);
St2 := '';
repeat
out:
read(Kbd,St1);
                                (* Read keyboard character pressed. *)
pass := (St1 in[#48..#57]);
                                (* Check for numeric keys only. *)

```

## APPENDIX C

```

If not Pass
  then
    if St1 = #13          (*      Check for enter key.      *)
      then
        goto Out1
      else
        goto Out;
    St2 := St2 + St1;
    GotoXY(X-Z,Y);
    Write(St2:Z);
  Out1:
  until St1 = #13;
  val(St2,Value,Code);
end; (* RightJustifyNumbers2 *)

(*-----*)

procedure RightJustifyNumbers4( X,Y,Z : integer; var Value : integer);

(* This procedure is used to read numbers in from the keyboard and to *)
(* display them on the screen right justified.                          *)

Label Out,Out1;

Var St1 : char;
    St4 : string[4];
    Pass : boolean;
    Code : integer;

begin (* RightJustifyNumbers4 *)
  gotoXY(X,Y);
  St4 := '';
  repeat
    out:
    read(Kbd,St1);          (* Read keyboard character pressed. *)
    pass := (St1 in[#48..#57]); (* Check for numeric keys only. *)
    If not Pass
      then
        if St1 = #13      (*      Check for enter key.      *)
          then
            goto Out1
          else
            goto Out;
        St4 := St4 + St1;
        GotoXY(X-Z,Y);
        Write(St4:Z);
      Out1:
      until St1 = #13;
      val(St4,Value,Code);
  end; (* RightJustifyNumbers4 *)

(*-----*)

procedure SysDateTime;

(* This procedure is used to check and change the system date and time. *)

Label Jump1,Jump2;

var Ch : char;
    Value : integer;

begin (* SysDateTime *)
  Jump1:
  MenuOutLine;
  gotoXY(28,4);
  write('«« SYSTEM DATE & TIME. »»');
  repeat
    window(5,7,75,17);      (* Set the size of the required work *)
                             (* area.                               *)
    clrscr;
    GetDate(Month,Day,Year,WeekDay);

```



```

gotoXY(17,3);
writeln('');
gotoXY(17,4);
writeln('');
gotoXY(17,5);
writeln(' Today is : ',Days[WeekDay]:9,Month:5,'/',Day:2,'/',Year:4,' ');
gotoXY(17,6);
writeln('');
gotoXY(17,7);
writeln(' Change the date ? (Y/N) ');
gotoXY(17,8);
writeln('');
gotoXY(17,9);
writeln('');
read(Kbd,ChangeDate); (* Read keyboard character pressed. *)
ChangeDate := upcase(ChangeDate);
(* Make sure variable is in upper case. *)
if (ChangeDate = 'Y') (* Check for the character 'Y'. *)
then
begin
clrscr;
gotoXY(20,4);
writeln('Enter new month (1-12) : ');
gotoXY(20,6);
writeln('Enter new day (1-31) : ');
gotoXY(20,8);
writeln('Enter new year (1980-2099) : ');
gotoXY(47,4);
CursorOn(true); (* Turn the cursor on. *)
RightJustifyNumbers2(47,4,2,Month);
RightJustifyNumbers2(45,6,2,Day);
RightJustifyNumbers4(53,8,4,Year);
CursorOn(false); (* Turn the cursor off. *)
SetDate(Month,Day,Year);
end;
until (ChangeDate <> 'Y');
if ChangeDate = #27 (* Check for escape sequence. *)
then
goto Jump2;
repeat
clrscr;
GetTime(Hour,Min,Sec);
gotoXY(23,3);
writeln('');
gotoXY(23,4);
writeln('');
gotoXY(23,5);
writeln(' The time is : ',Hour:4,':',Min:2,':',Sec:2,' ');
gotoXY(23,6);
writeln('');
gotoXY(23,7);
writeln(' Change the time ? (Y/N) ');
gotoXY(23,8);
writeln('');
gotoXY(23,9);
writeln('');
read(Kbd,ChangeTime); (* Read keyboard character pressed. *)
ChangeTime := upcase(ChangeTime);
(* Make sure variable is in upper case. *)
if (ChangeTime = 'Y') (* Check for the character 'Y'. *)
then
begin
clrscr;
gotoXY(20,4);
write('Enter new hour (0-24) : ');
gotoXY(20,6);
write('Enter new minute (0-60) : ');
gotoXY(20,8);
write('Enter new second (0-60) : ');
gotoXY(43,4);
CursorOn(true); (* Turn the cursor on. *)
RightJustifyNumbers2(46,4,2,Hour);

```

```

    RightJustifyNumbers2(48,6,2,Min);
    RightJustifyNumbers2(48,8,2,Sec);
    CursorOn(false);      (* Turn the cursor off. *)
    SetTime(Hour,Min,Sec);
end;
until (ChangeTime <> 'Y');
if ChangeTime = #27      (* Check for escape sequence. *)
then
    goto Jump2;
Jump2:
Get_Comm_Params;        (* Request serial port parameters. *)
if Initialize_Communications (* Initialize serial port. *)
then
    begin
        window(1,1,80,25); (* Set size of required work area to the *)
                                (* whole screen. *)
        Clrscr;
        MainMenu;
        SetBottomLine;
        window(11,16,70,21); (* Set the size of the required work *)
                                (* area. *)
        gotoXY(X,1);
    end
else
    goto Jump1;
end; (* SysDateTime *)

(*-----*)

procedure Setup;

(* This procedure makes it possible to set-up the communications parameters. *)

label Jump1;

var Ch : char;

begin (* Setup *)
    X := whereX;          (* Store the X-coordinate of the current *)
                            (* cursor position. *)

    Jump1:
    MenuOutline;
    gotoXY(31,4);
    write('««« SET-UP MENU »»»');
    repeat
        window(22,7,70,18); (* Set the size of the required work *)
                                (* area. *)
        gotoXY(1,2);
        writeln('COMMUNICATIONS PORT (' ,CP:4,' ) ----- (1)');
        writeln('BAUD RATE (' ,BR:4,' ) ----- (2)');
        writeln('PARITY (' ,PR:4,' ) ----- (3)');
        writeln('DATA BITS (' ,DB:4,' ) ----- (4)');
        writeln('STOP BITS (' ,SB:4,' ) ----- (5)');
        writeln('INPUT BUFFER SIZE (' ,IBS:4,' ) ----- (6)');
        writeln('OUTPUT BUFFER SIZE (' ,OBS:4,' ) ----- (7)');
        writeln('DO Xon/Xoff (' ,XX:4,' ) ----- (8)');
        writeln('DO CLEAR TO SEND (' ,CTS:4,' ) ----- (9)');
        writeln('DO DATA SET READY (' ,DSR:4,' ) ----- (0)');
        read(Kbd,Ch);
        case Ch of
            '1' : begin
                ReverseVideo(true);
                window(3,8,20,17); (* Set the size of the required work *)
                                        (* area. *)
                gotoXY(1,1);
                writeln(' |');
                writeln(' | COMMUNICATIONS |');
                writeln(' | PORT |');
                writeln(' |');
            end;
        end;
    until Ch = #27;
end;

```

```

writeln(' COM1 --- (1) ');
writeln(' COM2 --- (2) ');
writeln(' COM3 --- (3) ');
writeln(' COM4 --- (4) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : CP := 1;
  '2' : CP := 2;
  '3' : CP := 3;
  '4' : CP := 4;
end;
ReverseVideo(false);
Clrscr;
end;
'2' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                    (* area. *)

gotoXY(1,1);
writeln(' ');
writeln(' BAUD RATE ');
writeln(' ');
writeln(' 300 --- (1) ');
writeln(' 1200 --- (2) ');
writeln(' 2400 --- (3) ');
writeln(' 4800 --- (4) ');
writeln(' 9600 --- (5) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : BR := 300;
  '2' : BR := 1200;
  '3' : BR := 2400;
  '4' : BR := 4800;
  '5' : BR := 9600;
end;
ReverseVideo(false);
Clrscr;
end;
'3' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                    (* area. *)

gotoXY(1,1);
writeln(' ');
writeln(' PARITY ');
writeln(' ');
writeln(' ODD --- (1) ');
writeln(' EVEN --- (2) ');
writeln(' NONE --- (3) ');
writeln(' MARK --- (4) ');
writeln(' SPACE --- (5) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : PR := 'O';
  '2' : PR := 'E';
  '3' : PR := 'N';
  '4' : PR := 'M';
  '5' : PR := 'S';
end;
ReverseVideo(false);
Clrscr;
end;
'4' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                    (* area. *)

gotoXY(1,1);
writeln(' ');
writeln(' DATA BITS ');

```

```

writeln(' ');
writeln(' 5 BITS --- (1) ');
writeln(' 6 BITS --- (2) ');
writeln(' 7 BITS --- (3) ');
writeln(' 8 BITS --- (4) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : DB := 5;
  '2' : DB := 6;
  '3' : DB := 7;
  '4' : DB := 8;
end;
ReverseVideo(false);
Clrscr;
end;
'5' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                  (* area. *)
gotoXY(1,1);
writeln(' ');
writeln('  STOP BITS  ');
writeln(' ');
writeln(' 1 BIT --- (1) ');
writeln(' 2 BIT --- (2) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : SB := 1;
  '2' : SB := 2;
end;
ReverseVideo(false);
Clrscr;
end;
'6' : begin
ReverseVideo(true);
window(3,8,20,18); (* Set the size of the required work *)
                  (* area. *)
gotoXY(1,1);
writeln(' ');
writeln(' INPUT BUFFER ');
writeln('   SIZE      ');
writeln(' ');
writeln(' 512 --- (1) ');
writeln(' 1024 --- (2) ');
writeln(' 2048 --- (3) ');
writeln(' 4096 --- (4) ');
writeln(' 8192 --- (5) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : IBS := 512;
  '2' : IBS := 1024;
  '3' : IBS := 2048;
  '4' : IBS := 4096;
  '5' : IBS := 8192;
end;
ReverseVideo(false);
Clrscr;
end;
'7' : begin
ReverseVideo(true);
window(3,8,20,18); (* Set the size of the required work *)
                  (* area. *)
gotoXY(1,1);
writeln(' ');
writeln(' OUTPUT BUFFER ');
writeln('   SIZE      ');
writeln(' ');
writeln(' 512 --- (1) ');
writeln(' 1024 --- (2) ');

```

```

writeln(' 2048 --- (3) ');
writeln(' 4096 --- (4) ');
writeln(' 8192 --- (5) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : OBS := 512;
  '2' : OBS := 1024;
  '3' : OBS := 2048;
  '4' : OBS := 4096;
  '5' : OBS := 8192;
end;
ReverseVideo(false);
Clrscr;
end;
'8' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                    (* area. *)
gotoXY(1,1);
writeln(' ');
writeln(' DO Xon/Xoff ');
writeln(' ');
writeln(' YES --- (1) ');
writeln(' NO --- (2) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : XX := 'Y';
  '2' : XX := 'N';
end;
ReverseVideo(false);
Clrscr;
end;
'9' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                    (* area. *)
gotoXY(1,1);
writeln(' ');
writeln(' DO CTS ');
writeln(' ');
writeln(' YES --- (1) ');
writeln(' NO --- (2) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : CTS := 'Y';
  '2' : CTS := 'N';
end;
ReverseVideo(false);
Clrscr;
end;
'10' : begin
ReverseVideo(true);
window(3,8,20,17); (* Set the size of the required work *)
                    (* area. *)
gotoXY(1,1);
writeln(' ');
writeln(' DO DSR ');
writeln(' ');
writeln(' YES --- (1) ');
writeln(' NO --- (2) ');
writeln(' ');
read(Kbd,Ch);      (* Read keyboard character pressed. *)
case Ch of
  '1' : DSR := 'Y';
  '2' : DSR := 'N';
end;
ReverseVideo(false);
Clrscr;
end;

```

## APPENDIX C

```

end;
until Ch = #27;
Get_Comm_Params;          (* Request serial port parameters. *)
if Initialize_Communications (* Initialize serial port. *)
then
  begin
    window(1,1,80,25);    (* Set size of required work area to the *)
                          (* whole screen. *)
    Clrscr;
    MainMenu;
    SetBottomLine;
    window(11,16,70,21); (* Set the size of the required work *)
                          (* area. *)
    gotoXY(X,1);
  end
else
  goto Jump1;
end; (* Setup *)

(*-----*)

procedure Emulate_Dumb_Terminal;

(* This procedure is used to capture the data at the serial port. *)

var Kch      : char;
    Ch       : char;

begin (* Emulate_Dumb_Terminal *)
  repeat
    if KeyPressed          (* Monitor keyboard for a user requests. *)
    then
      begin
        read(Kbd,KCh);     (* Read keyboard character pressed. *)
        if (KCh = #27) and KeyPressed
        (* Check for escape sequences. *)
        then
          begin
            read(Kbd,KCh); (* Read keyboard character pressed. *)
            if KCh = #59 (* Check for <F1> key. *)
            then
              begin
                Async_Close(true);
                port[$3B8] := $29;
                (* Turn screen on. *)
                OfficeLayout;
                if (not Async_Open(Com_Port,Baud_Rate,Parity,Data_Bits,Stop_Bits))
                (* Try opening the serial port. *)
                then
                  writeln('Cannot open serial port.')
                  (* Serial port can't communicate due to *)
                  (* incorrect parameters. *)
                else
                  writeln('Serial port opened. ');
                  (* Serial port is ready to communicate. *)
                end;
              end
            else
              case KCh of
                '1' : begin
                  Async_Close(true);
                  port[$3B8] := $29;
                  (* Turn screen on. *)
                  Setup; (* If character pressed is '1', then *)
                  (* execute the Setup procedure. *)
                  if (not Async_Open(Com_Port,Baud_Rate,Parity,Data_Bits,Stop_Bits))
                  (* Try opening the serial port. *)
                  then
                    writeln('Cannot open serial port.')
                    (* Serial port can't communicate due to *)
                    (* incorrect parameters. *)

```

## APPENDIX C

```

else
  writeln('Serial port opened.');
```

(\* Serial port is ready to communicate. \*)

```

end;
'2' : begin
  Async_Close(true);
  port[$3B8] := $29;
  (* Turn screen on. *)
  PrintMenu;
  (* If character pressed is '2', then *)
  (* call the Print Menu procedure. *)
  if (not Async_Open(Com_Port,Baud_Rate,Parity,Data_Bits,Stop_Bits))
  (* Try opening the serial port. *)
  then
    writeln('Cannot open serial port.')
```

(\* Serial port can't communicate due to \*)

(\* incorrect parameters. \*)

```

  else
    writeln('Serial port opened.');
```

(\* Serial port is ready to communicate. \*)

```

end;
'3' : begin
  Async_Close(true);
  port[$3B8] := $29;
  (* Turn screen on. *)
  SysDateTime;
  (* If character pressed is '2', then *)
  (* call the Print Menu procedure. *)
  if (not Async_Open(Com_Port,Baud_Rate,Parity,Data_Bits,Stop_Bits))
  (* Try opening the serial port. *)
  then
    writeln('Cannot open serial port.')
```

(\* Serial port can't communicate due to \*)

(\* incorrect parameters. \*)

```

  else
    writeln('Serial port opened.');
```

(\* Serial port is ready to communicate. \*)

```

end;
'0' : begin
  port[$3B8] := $29;
  (* Turn screen on. *)
  Stop := true;
  (* If character pressed is '3', then *)
  (* terminate processing. *)
end;
#32 : begin
  if Blank
  then
    begin
      port[$3B8] := $21;
      (* Turn screen off. *)
      Blank := false;
    end
  else
    begin
      port[$3B8] := $29;
      (* Turn screen on. *)
      Blank := true;
    end;
  end;
end;
end;
if Async_Receive(Ch) (* Receive data at serial port. *)
then
  if (Ch <> chr(0)) (* Check if character received is valid. *)
  then
    begin
      if (Ch <> #10) and (Ch <> #13)
      (* Check for line feed and carriage *)
      (* return. *)
      then
        Cfstr := Cfstr + Ch;
```

APPENDIX C

```

                                (*      Build up a string until.      *)
write(Ch);
if (Ch = #13) and (length(Cfstr) < 80)
    (* Check for carriage return and that *)
    (* the strings length is less than 80 *)
    (* characters long.                  *)
then
begin
    X := whereX;
    (* Store the X-coordinate of the current *)
    (* cursor position.                    *)
    Y := whereY;
    (* Store the Y-coordinate of the current *)
    (* cursor position.                    *)
    GetData;
    Cfstr := '';
    (*      Reset string.                  *)
    gotoXY(X,Y);
end;
end;
until Stop;
end (* Emulate_Dumb_Terminal *);

(*-----*)
procedure ReadData;
(* This procedure is used to read data from serial port. *)
begin (* ReadData *)
    window(11,16,70,21);
    (* Set the size of the required work *)
    (* area.                             *)
    clrscr;
    Get_Comm_Params;
    (* Request serial port parameters. *)
    if Initialize_Communications
    (* Initialize serial port. *)
    then
    begin
        Emulate_Dumb_Terminal;
        (* Emulate dumb terminal. *)
        Finish_Communications;
        (* Close down serial port. *)
    end;
end (* ReadData *);

(*-----*)
procedure SetVar;
(* This procedure is used to set the relevant program variables at the *)
(* beginning of the program. *)
begin (* SetVar *)
    CP := 1;
    (* *)
    BR := 1200;
    (* *)
    PR := 'E';
    (* *)
    SB := 2;
    (* *)
    DB := 7;
    (* *)
    IBS := 2048;
    (* Set communications paramaters. *)
    OBS := 2048;
    (* *)
    XX := 'N';
    (* *)
    CTS := 'N';
    (* *)
    DSR := 'N';
    (* *)
    HW := 'N';
    (* *)
    FString := '';
    (* *)
    SString := '';
    (* Reset string. *)
    Cfstr := '';
    (* *)
    for I := 1 to 50 do
        PointIII := 0;
        (* Reset file pointers. *)
    end;
    Noc := 0;
    (* *)
    Raat := 0;
    (* *)
    TotRDur := 0;
    (* Reset program variables. *)
    Z := 0;
    (* *)
    Stop := false;
    (* *)
    Datum := '00/00/00';
    (* Reset program variables. *)
end;

```



## APPENDIX C

```

Ans := 0;          (* *)
Unans := 0;       (* *)
Blank := true;    (* *)
end; (* SetVar *)

(*-----*)

(* This is the main part of the program. *)

begin (* MonitorBTS *)
  SetVar;
  CursorOn(false);      (* Turn the cursor off. *)
  ClrScr;
  MainMenu;
  assign(Fil,'C:\TP3\DATA\BTS.DTA');
  (* Assign the file path and name to the *)
  (* variable name. *)
  rewrite(Fil);         (* Disk file is prepared for processing. *)
  assign(F,'C:\TP3\DATA\BTS_WED.DTA');
  (* Assign the file path and name to the *)
  (* variable name. *)
  reset(F);            (* Disk file is prepared for processing. *)
  seek(Fil,1);         (* File pointer is set to line 1 of file. *)
  window(11,16,70,21); (* Set the size of the required work *)
  (* area. *)
  ReadData;
  close(F);            (* The disk file is closed and the disk *)
  close(Fil);         (* directory is updated. *)
  window(1,1,80,25);  (* Set size of required work area to the *)
  (* whole screen. *)
  ClrScr;
  CursorOn(true);     (* Turn the cursor on. *)
end. (* MonitorBTS *)

```

APPENDIX C

```

(*-----*)
(* THIS IS AN INCLUDE FILE THAT TAKES CARE OF THE MENU INTERFACING WITH THE *)
(* INTERACTIVE USERS. *)
(*-----*)
(* Written by: A. M. Z. Molnar                               Version 1.0 *)
(*-----*)
(* Dated: November, 1991 *)
(*-----*)

procedure CursorOn(TurnOn : boolean);

(* This procedure is used to turn the cursor on the screen on or off. *)

Type regList = record
    AX, BX, CX, DX, BP, SI, DI, DS, ES, FLAGS : Integer;
end;

Var reg = regList;

begin (* CursorOn *)
    if TurnOn then
        if mem[0:$4491] = 7 then
            reg.CX := $0C00
        else
            reg.CX := $0607
        else
            reg.CX := $2000;
            reg.AX := $0100;
            intr($10, reg)
        end; (* CursorOn *)
    (*-----*)

procedure ReverseVideo(Status : boolean);

(* This procedure is used to toggle the screen display status to reverse *)
(* video or to normal, depending on the boolean argument received. *)

begin (* ReverseVideo *)
    if Status
    then
        begin
            TextColor(0);          (* Set text colour and text background *)
            TextBackGround(15);    (* to inverse. *)
        end
    else
        begin
            TextColor(15);         (* Set text colour and text background *)
            TextBackGround(0);     (* back to normal. *)
        end;
    end; (* ReverseVideo *)
    (*-----*)

procedure SetBottomLine;

(* This procedure is used to restore the variable information on the bottom *)
(* line, after returning to the main menu. *)

begin (* SetBottomLine *)
    gotoXY(20, 24);
    write(Datum);
    gotoXY(44, 24);
    write(Ans:4);
    gotoXY(66, 24);
    write(Unans:3);
end; (* SetBottomLine *)
(*-----*)

```

APPENDIX C

procedure NumBlock;

(\* This procedure is used as part of the main menu. It shows the various extensions. \*)

```
begin (* NumBlock *)
  gotoXY(1,5); write(' ');
  gotoXY(1,6); write(' 227 '); (* *)
  gotoXY(1,7); write(' 228 '); (* *)
  gotoXY(1,8); write(' 229 '); (* *)
  gotoXY(1,9); write(' 230 '); (* *)
  gotoXY(1,10); write(' 231 '); (* *)
  gotoXY(1,11); write(' 232 '); (* *)
  gotoXY(1,12); write(' 233 '); (* Extensions pertaining to *)
  gotoXY(1,13); write(' 234 '); (* 23 - 8880. *)
  gotoXY(1,14); write(' 235 '); (* *)
  gotoXY(1,15); write(' 236 '); (* *)
  gotoXY(1,16); write(' 245 '); (* *)
  gotoXY(1,17); write(' 247 '); (* *)
  gotoXY(1,18); write(' 248 '); (* *)
  gotoXY(1,19); write(' 249 '); (* *)
  gotoXY(1,20); write(' ');
  gotoXY(74,5); write(' ');
  gotoXY(74,6); write(' 225 '); (* *)
  gotoXY(74,7); write(' 226 '); (* *)
  gotoXY(74,8); write(' 237 '); (* *)
  gotoXY(74,9); write(' 238 '); (* *)
  gotoXY(74,10); write(' 239 '); (* *)
  gotoXY(74,11); write(' 240 '); (* *)
  gotoXY(74,12); write(' 241 '); (* Extensions pertaining to *)
  gotoXY(74,13); write(' 242 '); (* 22 - 2121. *)
  gotoXY(74,14); write(' 243 '); (* *)
  gotoXY(74,15); write(' 244 '); (* *)
  gotoXY(74,16); write(' 246 '); (* *)
  gotoXY(74,17); write(' 251 '); (* *)
  gotoXY(74,18); write(' 252 '); (* *)
  gotoXY(74,19); write(' 253 '); (* *)
  gotoXY(74,20); write(' ');
end; (* NumBlock *)
```

(\*-----\*)

procedure BottomLine;

(\* This procedure is part of the main menu. It lists the on line date as \*)  
 (\* well as the total amount of answered and unanswered calls. \*)

```
begin (* BottomLine *)
  gotoXY(9,23);
  write('_____');
  gotoXY(9,24);
  write(' Date : ANSWERED : UNANSWERED : ');
  gotoXY(9,25);
  write('_____');
end; (* BottomLine *)
```

(\*-----\*)

procedure MenuBlock;

(\* This procedure is part of the main menu. It gives the user a number of \*)  
 (\* options to choose from. \*)

```
begin (* MenuBlock *)
  gotoXY(9,1);
  write('_____');
  gotoXY(9,2);
  write(' TELEPHONIC ENQUIRIES MONITORING SYSTEM ');
  gotoXY(9,3);
  write('_____');
  gotoXY(9,4);
  write('_____');
  gotoXY(9,5);
  write('_____');
```

APPENDIX C

```

write('|');
gotoXY(9,6);
write('SET-UP UTILITY ----- (1) |');
gotoXY(9,7);
write('|');
gotoXY(9,8);
write('PRINT REPORTS ----- (2) |');
gotoXY(9,9);
write('|');
gotoXY(9,10);
write('SYSTEM DATE AND TIME ----- (3) |');
gotoXY(9,11);
write('|');
gotoXY(9,12);
write('QUIT ----- (0) |');
gotoXY(9,13);
write('|');
gotoXY(9,14);
write('L |');
gotoXY(10,14);
ReverseVideo(true);
write(' <F1> ---> Office lay-out and extension numbers. |');
ReverseVideo(false);
end; (* MenuBlock *)

(*-----*)

procedure DataBlock;

(* This procedure is part of the main menu. It is the block where the on *)
(* line data will scroll up and down in. *)

var A : integer;

begin (* DataBlock *)
gotoXY(9,15);
write('_____');
for A := 16 to 21 do
begin
gotoXY(9,A);
write('|');
gotoXY(72,A);
write('|');
gotoXY(9,22);
write('_____');
end;
end; (* DataBlock *)

(*-----*)

procedure MainMenu;

(* This procedure is used to combine a number of other procedures to form *)
(* the main menu. *)

begin (* MainMenu *)
NumBlock;
MenuBlock;
DataBlock;
BottomLine;
end; (* MainMenu *)

(*-----*)

procedure OfficeLayout;

(* This procedure is used as a reference guide to the user to indicate where *)
(* each extension is situated in the office. *)

var Ch : char;

begin (* OfficeLayout *)

```

APPENDIX C

```

X := whereX;          (* Store the X-coordinate of the current *)
                      (* cursor position. *)
window(1,1,80,25);   (* Set size of required work area to the *)
                      (* whole screen. *)

ClrScr;
write('');
for I := 2 to 23 do
begin
  gotoXY(1,1);
  write('');
  gotoXY(79,1);
  write('');
end;
gotoXY(1,24);
write('');
gotoXY(8,11);
window(8,11,16,14);  (* Set the size of the required work *)
                      (* area. *)

writeln('FRONT');
writeln(' OF');
writeln('OFFICE');
window(1,1,80,25);  (* Set size of required work area to the *)
                      (* whole screen. *)

gotoXY(1,25);
ReverseVideo(true);
write(' To return to main menu press <ESC> key. ');
ReverseVideo(false);
gotoXY(20,3);
window(20,3,75,22); (* Set the size of the required work *)
                      (* area. *)

repeat
  writeln('');
  writeln(' 2 2 2 2 2 2 2 2 ');
  writeln(' 2 2 2 3 3 3 4 4 ');
  writeln(' 7 8 9 0 1 2 7 8 ');
  writeln('');
  writeln('');
  writeln(' 2 2 2 2 2 2 2 2 ');
  writeln(' 4 4 5 3 3 3 4 3 ');
  writeln(' 3 4 3 5 4 3 9 6 ');
  writeln('');
  writeln('');
  writeln(' 2 2 2 2 2 2 2 2 ');
  writeln(' 3 3 3 4 4 4 5 5 ');
  writeln(' 7 8 9 0 1 2 1 2 ');
  writeln('');
  read(Kbd,Ch);      (* Read keyboard character pressed. *)
  until Ch = #27;    (* Check for escape sequence. *)
  window(1,1,80,25); (* Set size of required work area to the *)
                      (* whole screen. *)

ClrScr;
MainMenu;
SetBottomLine;
window(11,16,70,21); (* Set the size of the required work *)
                      (* area. *)

gotoXY(X,1);
end; (* OfficeLayout *)

(*-----*)

procedure MenuOutline;

(* This procedure is used to determine which reports are required. *)

begin (* MenuOutline *)
  X := whereX;      (* Store the X-coordinate of the current *)
                    (* cursor position. *)
  window(1,1,80,25); (* Set size of required work area to the *)

```

## APPENDIX C

```

(* whole screen. *)
ClrScr;
gotoXY(1,3);
writeln(' ');
writeln(' ');
writeln(' ');
writeln(' ');
for I := 1 to 12 do
  writeln(' | ');
writeln(' | ');
writeln(' | ');
writeln(' | ');
gotoXY(8,20);
ReverseVideo(true);
write(' To return to main menu press <ESC> key. ');
ReverseVideo(false);
end; (* MenuOutline *)

```

## APPENDIX C

```

(*-----*)
(* THIS IS AN INCLUDE FILE THAT TAKES CARE OF THE PRINTING INTERFACE WITH *)
(* THE INTERACTIVE USERS. *)
(* *)
(* Written by: A. M. Z. Molnar Version 1.0 *)
(* *)
(* Dated: November, 1991 *)
(* *)
(*-----*)

function Con(Time : str8) : real;

(* This function is used to convert a string that represents time, into a *)
(* real number that can be used in various manipulations. *)

var Hour,Min,Sec : integer;

begin (* Con *)
  val(copy(Time,1,2),Hour,Err);
  val(copy(Time,4,2),Min,Err);
  val(copy(Time,7,2),Sec,Err);
  Con := (Hour * 60) + Min + (Sec / 60);
end; (* Con *)

(*-----*)

function ReCon(Time : real) : str8;

(* This function is used to re-convert a real number that represents a time *)
(* into a string for output purposes. *)

var Hour,Min,Sec : string[2];

begin (* ReCon *)
  str(int(Time / 60):2:0,Hour);
  str(int(frac(Time / 60) * 60):2:0,Min);
  str(frac(Time) * 60:2:0,Sec);
  if Min[1] = ' '
  then
    Min[1] := '0';
  if Sec[1] = ' '
  then
    Sec[1] := '0';
  ReCon := Hour + ':' + Min + ':' + Sec;
end; (* ReCon *)

(*-----*)

function PrinterStatus(WhichPrinter : integer) : integer;

type Registers = record case integer of
  1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : integer);
  2 : (AL,AH,BL,BH,CL,CH,DL,DH : byte);
end;

var Regs : Registers;

begin
  Regs.AH := 2;
  Regs.AL := 0;
  Regs.DX := WhichPrinter;
  Intr(23,Regs);
  PrinterStatus := Regs.AH;
end;

(*-----*)

```

APPENDIX C

```

procedure PrinterError(StatusByte : integer;
    var ErrorCode : integer;
    var ErrorReturn : str20);

var X : integer;

const Error : array[0..2] of string[20] = ('    No error      ', 'Printer not ready',
    '    Printer off   ');

begin (* PrinterError *)
    if StatusByte in [144,146,216,223]
        (*          Check for no errors.          *)
    then
        ErrorCode := 0;
    if StatusByte in [24,80,87] (*          Check if printer is on line.          *)
    then
        ErrorCode := 1;
    if StatusByte in [120,135,200] (*          Check if printer is switched on.          *)
    then
        ErrorCode := 2;
    ErrorReturn := Error[ErrorCode];
        (*          Set error message.          *)
end; (* PrinterError *)

(*-----*)

procedure TestPrinter(var YesNo : boolean);

var B,X : integer;
    Error : string[20];

begin (* TestPrinter *)
    YesNo := true;
    B := PrinterStatus(0);
    PrinterError(B,X,Error);
    if X <> 0
    then
        begin
            YesNo := false;
            gotoXY(17,11);
            ReverseVideo(true);
            write(' ');
            gotoXY(17,12);
            write(' <<<< PRINTER ERROR = ',Error,' >>>> ');
            gotoXY(17,13);
            write(' ');
            delay(5000);
            ReverseVideo(false);
        end;
end; (* TestPrinter *)

(*-----*)

procedure GetPrintInfo;

(* This procedure is used to obtain the necessary data from the disk file *)
(* to be used for the printed reports. *)

begin (* GetPrintInfo *)
    reset(Fil);
    seek(Fil,1);
    for R := 220 to 270 do
        for S := 7 to 16 do
            for T := 1 to 5 do
                InfoExt[R,S,T] := 0;
    for R := 1 to 30 do
        for S := 7 to 16 do
            for T := 1 to 3 do
                InfoLn[R,S,T] := 0;
    for R := 1 to 30 do
        for S := 7 to 16 do
            for T := 1 to 2 do
                (* Disk file is prepared for processing. *)
                (* File pointer is set to line 1 of file. *)
                (*          *)
                (*          Reset the extension information          *)
                (*          variables.          *)
                (*          *)
                (*          Reset the line information          *)
                (*          variables.          *)
                (*          *)
                (*          Reset the unanswered information          *)
                (*          variables.          *)

```



```

        InfoUna[R,S,T] := 0;      (*)
for S := 1 to 255 do
    Flag[S] := 0;              (* Reset trunk line and extension flags. *)
gotoXY(15,10);
ReverseVideo(true);
write('_____');
gotoXY(15,11);
write('    <<< PROCESSING DATA FOR PRINTED REPORT: >>>    ');
gotoXY(15,12);
write('_____');
repeat
    read(Fil,Data);           (* Read data from data file. *)
gotoXY(15,13);
with Data do write(' | ',Date,' | ',Time,' | ',Rec,' | ',Ext,' | ',RDur,' | ',CDur,' | ',Ln:2,' | ');
gotoXY(15,14);
write('_____');
val(copy(Data.Time,1,2),Tim,Err);
                                (* Make the variable Tim equal to the *)
                                (* integer value of the hour portion *)
                                (* of Data.Time (Time of day). *)
val(Data.Ext,Exten,Err);        (* Make the variable Exten equal to the *)
                                (* integer value of Data.Ext (Extension *)
                                (* number). *)
if Err = 0                      (* Check if record is answered (Err = 0) *)
                                (* or unanswered (Err = 1). *)
then
begin
    InfoExt[Exten,Tim,1] := InfoExt[Exten,Tim,1] + 1;
                                (* Count amount of calls answered per *)
                                (* extension per hour. *)
    InfoExt[Exten,Tim,2] := InfoExt[Exten,Tim,2] + Con(Data.CDur);
                                (* Total speech time per extension per *)
                                (* hour. *)
    InfoExt[Exten,Tim,3] := InfoExt[Exten,Tim,3] + Con(Data.RDur);
                                (* Total ringing time per extension *)
                                (* per hour. *)
    if InfoExt[Exten,Tim,4] < Con(Data.CDur)
    then
        InfoExt[Exten,Tim,4] := Con(Data.CDur);
                                (* Longest call per extension per hour. *)
    if (InfoExt[Exten,Tim,5] <> 0) and (InfoExt[Exten,Tim,5] < Con(Data.CDur))
    then
        InfoExt[Exten,Tim,5] := InfoExt[Exten,Tim,5]
    else
        InfoExt[Exten,Tim,5] := Con(Data.CDur);
                                (* Shortest call per extension per hour. *)
    InfoLn[Data.Ln,Tim,1] := InfoLn[Data.Ln,Tim,1] + 1;
                                (* Count amount of calls per line per *)
                                (* hour. *)
    InfoLn[Data.Ln,Tim,2] := InfoLn[Data.Ln,Tim,2] + Con(Data.CDur);
                                (* Total speech time per line per hour. *)
    InfoLn[Data.Ln,Tim,3] := InfoLn[Data.Ln,Tim,3] + Con(Data.RDur);
                                (* Total ringing time per line per hour. *)
end
else
begin
    InfoUna[Data.Ln,Tim,1] := InfoUna[Data.Ln,Tim,1] + 1;
                                (* Count amount of unanswered calls per *)
                                (* line per hour. *)
    InfoUna[Data.Ln,Tim,2] := InfoUna[Data.Ln,Tim,2] + Con(Data.RDur);
                                (* Total ringing time of unanswered *)
                                (* calls per line per hour. *)
end;
for S := 225 to 255 do
    for R := 7 to 16 do
        if InfoExt[S,R,1] <> 0
        then
            Flag[S] := 1;
        end;
for S := 1 to 30 do
    for R := 7 to 16 do
        if (InfoLn[S,R,1] <> 0) or (InfoUna[S,R,1] <> 0)
        then
            (* Set flags to determine which trunk *)

```

APPENDIX C

```

Flag[SI] := 1;      (* lines are used each hour.      *)
until eof(Fil);
ReverseVideo(false);
ClrScr;
end; (* GetPrintInfo *)

(*-----*)

procedure ExtRep;

(* This procedure is used to produce a printed report regarding each *)
(* individual extensions hourly progress.                          *)

var TotAns,TotDur,AvgDur,Long,Short,Idle : real;

begin (* ExtRep *)
  for S := 225 to 255 do
    if Flag[SI] = 1 then      (* Check which extensions answered calls. *)
      begin
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        for R := 7 to 8 do
          begin
            if InfoExt[S,R,2] > 60
              then
                Idle := 0
              else
                Idle := (60 - InfoExt[S,R,2]);
            if InfoExt[S,R,1] <> 0
              then
                begin
                  write(lst, ' | 0',R,':00 - 0',R+1,':00',InfoExt[S,R,1]:10:0,ReCon(InfoExt[S,R,2]):10);
                  write(lst,ReCon(InfoExt[S,R,2]/InfoExt[S,R,1]):10,copy(ReCon(Idle),4,5):7);
                  writeln(lst,ReCon(InfoExt[S,R,4]):10,ReCon(InfoExt[S,R,5]):10,' | ');
                end;
              end;
            (* For the period between 07:00 and 09:00 print on a hourly basis the amount *)
            (* of calls answered, the total speech duration, the average speech duration *)
            (* , the idle time, the longest call and the shortest call for the relevant *)
            (* extension. *)
            if InfoExt[S,9,2] > 60
              then
                Idle := 0
              else
                Idle := (60 - InfoExt[S,9,2]);
            if InfoExt[S,9,1] <> 0
              then
                begin
                  write(lst, ' | 09:00 - 10:00',InfoExt[S,9,1]:10:0,ReCon(InfoExt[S,9,2]):10);
                  write(lst,ReCon(InfoExt[S,9,2]/InfoExt[S,9,1]):10,copy(ReCon(Idle),4,5):7);
                  writeln(lst,ReCon(InfoExt[S,9,4]):10,ReCon(InfoExt[S,9,5]):10,' | ');
                end;
            (* For the period between 09:00 and 10:00 print on a hourly basis the amount *)
            (* of calls answered, the total speech duration, the average speech duration *)
            (* , the idle time, the longest call and the shortest call for the relevant *)
            (* extension. *)
            for R := 10 to 16 do
              begin
                if InfoExt[S,R,2] > 60
                  then
                    Idle := 0
                  else
                    Idle := (60 - InfoExt[S,R,2]);
                if InfoExt[S,R,1] <> 0
                  then
                    begin

```

EXTENSION : 'S:3,'	DATE: 'DATUM,'					
Time	Amount Answered	Total Duration	Average Duration	Idle Time	Longest Call	Shortest Call

```

          end;
        end;
      end;
    end;
  end;
end;

```

APPENDIX C

```

write(lst, '| ',R,':00 - ',R+1,':00',InfoExt[S,R,1]:10:0,ReCon(InfoExt[S,R,2]):10);
write(lst,ReCon(InfoExt[S,R,2]/InfoExt[S,R,1]):10,copy(ReCon(Idle),4,5):7);
write(lst,ReCon(InfoExt[S,R,4]):10,ReCon(InfoExt[S,R,5]):10,' |');
end;
end;
(* For the period between 10:00 and 17:00 print on an hourly basis the amount *)
(* of calls answered, the total speech duration, the average speech duration *)
(* , the idle time, the longest call and the shortest call for the relevant *)
(* extension. *)
write(lst,'
write(lst,'
TotAns := 0; (* *)
TotCDur := 0; (* *)
AvgCDur := 0; (* Reset variables. *)
Long := 0; (* *)
Short := 0; (* *)
for R := 7 to 16 do
begin
TotAns := TotAns + InfoExt[S,R,1];
(* Total number of calls answered for a *)
(* day per relevant extension. *)
TotCDur := TotCDur + InfoExt[S,R,2];
(* Total speech time for a day per *)
(* relevant extension. *)
if Long < InfoExt[S,R,4]
then
Long := InfoExt[S,R,4];
(* Longest call for a day per relevant *)
(* extension. *)
if (Short < 0) and (Short < InfoExt[S,R,5])
then
Short := Short
else
Short := InfoExt[S,R,5];
(* Shortest call for a day per relevant *)
(* extension. *)
end;
AvgCDur := TotCDur / TotAns;
(* Average speech time per call for a *)
(* day per relevant extension. *)
write(lst,'| ',TotAns:23:0,ReCon(TotCDur):10,ReCon(AvgCDur):10);
write(lst,ReCon(Long):17,ReCon(Short):10,' |');
(* Print the daily totals per relevant extension of the total amount of calls *)
(* answered, the total speech time, the average speech time, the longest *)
(* call and the shortest call. *)
write(lst,'
write(lst,'
write(lst,'
write(lst,#12); (* Add a form feed after the print-out. *)
end;
end; (* ExtRep *)

(*-----*)

procedure ManRep;

(* This procedure is used to produce a printed report regarding the *)
(* switch board as a whole. *)

var CountExt,LExt,SExt : array[7..16] of integer;
Long,Short,AvgCDur,TotCDur,TotAns,TotUna : array[7..16] of real;
TotalAns,TotalUna,TotalCDur : real;
TotalLong,TotalShort : real;
TotalLExt,TotalSExt : integer;

begin (* ManRep *)
TotalAns := 0; (* *)
TotalUna := 0; (* *)
TotalCDur := 0; (* *)
TotalLong := 0; (* Reset variables. *)
TotalShort := 0; (* *)
TotalLExt := 0; (* *)

```

# APPENDIX C

```

TotalSExt := 0;
for R := 7 to 16 do
begin
    CountExt[R] := 0;
    TotAns[R] := 0;
    TotUna[R] := 0;
    TotCDur[R] := 0;
    AvgCDur[R] := 0;
    Long[R] := 0;
    Short[R] := 0;
    LExt[R] := 0;
    SExt[R] := 0;
end;
for R := 7 to 16 do
for S := 225 to 255 do
if InfoExt[S,R,1] <> 0
then
begin
    CountExt[R] := CountExt[R] + 1;
    TotAns[R] := TotAns[R] + InfoExt[S,R,1];
    TotCDur[R] := TotCDur[R] + InfoExt[S,R,2];
    AvgCDur[R] := TotCDur[R]/TotAns[R];
    if Long[R] < InfoExt[S,R,4]
    then
        begin
            LExt[R] := S;
            Long[R] := InfoExt[S,R,4];
        end;
    if (Short[R] <> 0) and (Short[R] < InfoExt[S,R,5])
    then
        Short[R] := Short[R]
    else
        begin
            SExt[R] := S;
            Short[R] := InfoExt[S,R,5];
        end;
end;
for R := 7 to 16 do
begin
    for S := 1 to 30 do
    if InfoUna[S,R,1] <> 0
    then
        TotUna[R] := TotUna[R] + InfoUna[S,R,1];
end;
writeln(lst, '
writeln(lst, '
writeln(lst, '
writeln(lst, '
writeln(lst, '
writeln(lst, '
writeln(lst, '
writeln(lst, '
writeln(lst, '
for R := 7 to 16 do
if CountExt[R] <> 0
then
begin
write(lst, ' ',R:2, ' - ',R+1:2, CountExt[R]:6, TotAns[R]:10:0, TotUna[R]:12:0);
write(lst, copy(ReCon(AvgCDur[R]),4,5):10, ' (',LExt[R]:3, ')', copy(ReCon(Long[R]),4,5):6);
writeln(lst, ' (',SExt[R]:3, ')', copy(ReCon(Short[R]),4,5):6, ' ');
end;

```

APPENDIX C

```

(* For the period between 07:00 and 17:00 print on a hourly basis the number *)
(* of extensions used, the number of calls answered, the number of calls *)
(* unanswered, the average speech time per call, the longest call and the *)
(* extension that made it and the shortest call and the extension that made *)
(* it. *)
writeln(lst,'
writeln(lst,'
for R := 7 to 16 do
begin
  TotalAns := TotalAns + TotAns[R];
  (* Total number of calls answered per *)
  (* day. *)
  TotalUna := TotalUna + TotUna[R];
  (* Total number of calls unanswered per *)
  (* day. *)
  TotalCDur := TotalCDur + TotCDur[R];
  (* Average speech time per call per day. *)
  if TotalLong < Long[R]
  then
  begin
    TotalLong := Long[R];
    TotalLExt := LExt[R];
  end;
  (* Longest call per day and which *)
  (* extension made it. *)
  if (TotalShort <> 0) and (TotalShort < Short[R])
  then
  begin
    TotalShort := TotalShort;
    TotalSExt := TotalSExt;
  end
  else
  begin
    TotalShort := Short[R];
    TotalSExt := SExt[R];
  end;
  (* Shortest call per day and which *)
  (* extension made it. *)
end;
writeln(lst,'|',TotalAns:24:0,TotalUna:12:0,copy(ReCon(TotalCDur/TotalAns),4,5):10);
writeln(lst,copy(ReCon(TotalLong),4,5):15,copy(ReCon(TotalShort),4,5):12,'|');
(* Print the daily totals for the whole switch board of the total amount of *)
(* calls answered, the total amount of calls unanswered, the average speech *)
(* time per call, the longest call and the shortest call. *)
writeln(lst,'
writeln(lst,'
writeln(lst,'
write(lst,#12);
(* Add a form feed after the print-out. *)
end; (* ManRep *)

(*-----*)

procedure LnRep;

(* This procedure is used to produce a printed report regarding each *)
(* individual trunk line's hourly progress. *)

var TotalAns,TotalRCDur,TotalUna,TotalRDur,TotalDur : real;

begin (* LnRep *)
  for S := 1 to 30 do
    if Flag[S] = 1 (* Check which trunk lines are used. *)
    then
      begin
        TotalAns := 0; (* *)
        TotalRCDur := 0; (* *)
        TotalUna := 0; (* Reset variables. *)
        TotalRDur := 0; (* *)
        TotalDur := 0; (* *)
        writeln(lst,'
        writeln(lst,'
        writeln(lst,'
        TRUNK LINE : ',S:2,' DATE: ',DATUM,'

```

APPENDIX C

```

writeln(lst,'
writeln(lst,'
writeln(lst,'
writeln(lst,'
writeln(lst,'
for R := 7 to 8 do
  if (InfoLn[S,R,1] <> 0) or (InfoUna[S,R,1] <> 0)
  then
    begin
      write(lst,' | 0',R,':00 - 0',R+1,':00',InfoLn[S,R,1]:7:0);
      write(lst,Recon(InfoLn[S,R,2]+InfoLn[S,R,3]):13,InfoUna[S,R,1]:8:0);
      write(lst,Recon(InfoUna[S,R,2]):14);
      writeln(lst,Recon(InfoLn[S,R,2]+InfoLn[S,R,3]+InfoUna[S,R,2]):13,' |');
    end;
  (* For the period between 07:00 and 09:00 print on a hourly basis the amount *)
  (* of calls answered, the ringing and call duration, the unanswered calls, *)
  (* the ringing duration and the total duration the relevant trunk lines are *)
  (* busy. *)
  if (InfoLn[S,9,1] <> 0) or (InfoUna[S,9,1] <> 0)
  then
    begin
      write(lst,' | 09:00 - 10:00',InfoLn[S,9,1]:7:0);
      write(lst,Recon(InfoLn[S,9,2]+InfoLn[S,9,3]):13,InfoUna[S,9,1]:8:0);
      write(lst,Recon(InfoUna[S,9,2]):14);
      writeln(lst,Recon(InfoLn[S,9,2]+InfoLn[S,9,3]+InfoUna[S,9,2]):13,' |');
    end;
  (* For the period between 09:00 and 10:00 print on a hourly basis the amount *)
  (* of calls answered, the ringing and call duration, the unanswered calls, *)
  (* the ringing duration and the total duration the relevant trunk lines are *)
  (* busy. *)
  for R := 10 to 16 do
    if (InfoLn[S,R,1] <> 0) or (InfoUna[S,R,1] <> 0)
    then
      begin
        write(lst,' | ',R,':00 - ',R+1,':00',InfoLn[S,R,1]:7:0);
        write(lst,Recon(InfoLn[S,R,2]+InfoLn[S,R,3]):13,InfoUna[S,R,1]:8:0);
        write(lst,Recon(InfoUna[S,R,2]):14);
        writeln(lst,Recon(InfoLn[S,R,2]+InfoLn[S,R,3]+InfoUna[S,R,2]):13,' |');
      end;
    (* For the period between 10:00 and 17:00 print on a hourly basis the amount *)
    (* of calls answered, the ringing and call duration, the unanswered calls, *)
    (* the ringing duration and the total duration the relevant trunk lines are *)
    (* busy. *)
    writeln(lst,'
    writeln(lst,'
    for R := 7 to 16 do
      begin
        TotalAns := TotalAns + InfoLn[S,R,1];
          (* Total number of calls answered per *)
          (* day per relevant trunk line. *)
        TotalRCDur := TotalRCDur + InfoLn[S,R,2] + InfoLn[S,R,3];
          (* Total ringing and call duration for *)
          (* answered calls per day per relevant *)
          (* trunk line. *)
        TotalUna := TotalUna + InfoUna[S,R,1];
          (* Total number of unanswered calls per *)
          (* day per relevant trunk line. *)
        TotalRDur := TotalRDur + InfoUna[S,R,2];
          (* Total ringing duration for unanswered *)
          (* calls per day per relevant trunk line. *)
        TotalDur := TotalRCDur + TotalRDur;
          (* Total duration per day that relevant *)
          (* trunk line is busy. *)
      end;
    write(lst,' | ',TotalAns:22:0,ReCon(TotalRCDur):13,TotalUna:8:0);
    writeln(lst,ReCon(TotalRDur):14,ReCon(TotalDur):13,' |');
  (* Print the daily totals per relevant trunk line of the total number of *)
  (* calls answered, the total ringing and call duration, the total number *)
  (* of unanswered calls, the total ringing duration and the total duration *)
  (* the line is busy. *)
  writeln(lst,'
  writeln(lst,'

```

```

        writeln(lst,'
        write(lst,#12);      (* Add a form feed after the print-out. *)
    end;
end; (* LnRep *)

(*-----*)

procedure ExtAnsRep;

(* This procedure is used to produce a printed report of the average *)
(* daily answering times of both the operators as well as the switch *)
(* board as a whole. *)

var TotRDur,TotAns,AvgRDur,TotCDur,AvgCDur : real;
    TotalAns,TotalRDur,TotalCDur : real;

begin
    writeln(lst,'
    writeln(lst,'
    writeln(lst,'
    writeln(lst,'
    writeln(lst,'
    writeln(lst,'
    writeln(lst,'
    for S := 225 to 255 do
        begin
            TotRDur := 0;          (*
            TotAns := 0;          (*
            AvgRDur := 0;         (*
            TotCDur := 0;         (*
            AvgCDur := 0;         (*
            TotalAns := 0;        (*
            TotalRDur := 0;       (*
            TotalCDur := 0;       (*
            for R := 7 to 16 do
                begin
                    TotRDur := TotRDur + InfoExt[S,R,3];
                    (* Total ringing duration per extension. *)
                    TotAns := TotAns + InfoExt[S,R,1];
                    (* Total amount of calls answered per *)
                    (* extension. *)
                    TotCDur := TotCDur + InfoExt[S,R,2];
                    (* Total speech duration per extension. *)
                end;
                if TotAns <> 0
                then
                    begin
                        AvgRDur := TotRDur / TotAns;
                        (* Average ringing duration per *)
                        (* extension. *)
                        AvgCDur := TotCDur / TotAns;
                        (* Average speech duration per *)
                        (* extension. *)
                        writeln(lst,'|',S:8,TotAns:15:0,ReCon(AvgRDur):22,ReCon(AvgCDur):22,'|');
                        (* Print the daily totals for each extension of the amount of calls answered *)
                        (* ,the average ringing duration and the average speech time. *)
                    end;
                end;
            for R := 7 to 16 do
                for S := 225 to 255 do
                    begin
                        TotalAns := TotalAns + InfoExt[S,R,1];
                        (* Total number of calls answered by the *)
                        (* switch board as a whole. *)
                        TotalRDur := TotalRDur + InfoExt[S,R,3];
                        (* Total ringing duration as seen by the *)
                        (* switch board as a whole. *)
                        TotalCDur := TotalCDur + InfoExt[S,R,2];
                        (* Total speech duration as seen by the *)
                        (* switch board as a whole. *)
                    end;
                end;
            writeln(lst,'| Total',TotalAns:14:0,ReCon(TotalRDur/TotalAns):22,ReCon(TotalCDur/TotalAns):22,'
    );
end;

```

APPENDIX C

```

(* Print the daily totals for the whole switch board of the total amount of *)
(* calls answered, the average ringing time per call and the average speech *)
(* time per call. *)
writeln(lst,' ');
writeln(lst,' ');
write(lst,#12); (* Add a form feed after the print-out. *)
end;

(*-----*)

procedure PrintMenu;

(* This procedure is used to select the required print-out. *)

label Jump1,Jump2;

var Ch : char;
    YesNo : boolean;

begin (* PrintMenu *)
  Jump1:
  MenuOutLine;
  gotoXY(31,4);
  write('*** PRINT MENU. ***');
  window(25,7,65,18); (* Set the size of the required work *)
                      (* area. *)

  gotoXY(1,2);
  writeln('SWITCH BOARD STATISTICS ----- (1)');
  writeln;
  writeln('INDIVIDUAL EXTENSIONS ----- (2)');
  writeln;
  writeln('TRUNK LINE REPORTS ----- (3)');
  writeln;
  writeln('AVERAGE ANSWERING TIMES ----- (4)');
  writeln;
  writeln('ALL REPORTS ----- (5)');
  window(1,1,80,25); (* Set size of required work area to the *)
                    (* whole screen. *)
  read(Kbd,Ch); (* Read keyboard character pressed. *)
  if Ch = #27 (* Check for escape sequences. *)
  then
    goto Jump2
  else
    begin
      case Ch of
        '1' : begin
          TestPrinter(YesNo);
          if YesNo
          then
            begin
              write(lst,#27#67#33);
                (* Set-up page size on printer. *)
              GetPrintInfo;
              ManRep; (* If character pressed is '1', then *)
                (* print the management report. *)
            end;
          end;
        '2' : begin
          TestPrinter(YesNo);
          if YesNo
          then
            begin
              write(lst,#27#67#33);
                (* Set-up page size on printer. *)
              GetPrintInfo;
              ExtRep; (* If character pressed is '2', then *)
                (* print the extension report. *)
            end;
          end;
        '3' : begin
          TestPrinter(YesNo);
          if YesNo

```



APPENDIX C

```

    then
    begin
        write(lst,#27#67#33);
            (*      Set-up page size on printer.      *)
        GetPrintInfo;
        LnRep; (* If character pressed is '3', then *)
            (* print the trunk line report. *)
    end;
end;
'4' : begin
    TestPrinter(YesNo);
    if YesNo
    then
    begin
        write(lst,#27#67#33);
            (*      Set-up page size on printer.      *)
        GetPrintInfo;
        ExtAnsRep;
            (* If character pressed is '4', then *)
            (* print the answering times report. *)
    end;
end;
'5' : begin
    TestPrinter(YesNo);
    if YesNo
    then
    begin
        write(lst,#27#67#33);
            (*      Set-up page size on printer.      *)
        GetPrintInfo;
        ManRep; (* If character pressed is '5', then *)
        ExtRep; (* print the management,extension,trunk *)
        LnRep; (* line and answering times reports. *)
        ExtAnsRep;
    end;
end;
else
    goto Jump1;
end;
end;
Jump2:
ClrScr;
MainMenu;
SetBottomLine;
window(11,16,70,21); (* Set the size of the required work *)
                        (* area. *)
gotoXY(X,1);
end; (* PrintMenu *)

(*-----*)

```