MICROCOMPUTER-BASED TWO LEVEL SUPERVISORY CONTROL AND DATA ACQUISITION SYSTEM

A DISSERTATION

Submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF ENGINEERING

in

ELECTRICAL ENGINEERING

By

ANUPAMA SINGHAL

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE–247 667 INDIA

FEBRUARY, 1990

CANDIDATE'S DECLARATION

I hereby certify that the work presented in this dissertation entitled," MICROCOMPUTER-BASED TWO LEVEL SUPERVISORY CONTROL AND DATA ACQUISITION SYSTEM" in partial fulfilment of the requirements for the award of degree of "MASTER OF ENGINEERING" (ELECTRICAL) with specialization in "MEASUREMENT AND INSTRUMENTATION" submitted in the Department of Electrical Engineering, University of Roorkee, Roorkee(India) is an authentic record of my own work carried out during the period of July 1989 to February 1990, under the supervision of Dr. H.K. Verma, Professor, Electrical Engineering Department, University of Roorkee, Roorkee. India.

The matter embodied in this dissertation has not submitted by me for any other degree or diploma.

Anupama Singhal

Dated: 26th Feb, 1990

(Anupama Singhal)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Dated: 26th Feb, 1990.

Dr. H.K. VERMA
Professor
Electrical Engg. Deptt.
University of Roorkee
ROORKEE, U.P. INDIA

A C K N O W L E D G E M E N T

The author considers it a pleasent duty to express her heartiest appreciation and gratitude to Dr. H.K. VERMA, Professor, Department of Electrical Engineering, University of Roorkee, Roorkee for his keen interest, invaluable and painstaking excellent guidance, continuous calms and throughout advice during each and everyphase of present work.

I, owe, my sincere thanks to Dr. R.B. SAXENA, Professor and Head, Department of Electrical Engineering for providing all the facilities.

I also take the privilege of thanking Mr. G.R.Verma, Mr.C.P. Kansal and Mr. S.D. Mishra who took keen interest and great pleasure to enhance my enthusiasm for this dissertation.

I am again greatly thankful to Mr. M.K. Vasantha, Professor in Electrical Engineering, Dr.J.D. Sharma, Professor in Electrical Engineering who did their best to co-ordinate me in successful completion of this work.

I also sincerely thank my parents, my in-laws and my husband Mr. Pankaj Agarwal, who constantly inspired me even though from a distance, during this work.

Anupama Singhal

Anupama Singhal

S Y N O P S I S

The industrial systems in which there are distinct units of operation, often unattended and which require supervision from a central facility, use supervisory control and data acquisition (SCADA), where in control centre operators can monitor and control the devices at remote place. Functionally, SCADA consists in the acquisition of data from the controlled system, processing the data, displaying the processed data ata central computer system and giving the commands to operate devices at remote places from the control centre.

A review of the developments in remote control, telemetry and supervisory control techniques since the World War II has been carried out. trends in communication with respect to the impact of the development of transistors, large automatic tracking antennas, phase locked FM detector and technologically superior communication mediams, are also looked at. The use of computers in SCADA is overviewed.

A two level SCADA system has been developed in this project. For the hardware of the remote terminal unit (RTU) of this system, Intel 8085 micro-processor based card cage micro-computer system is used. With the help of different modules of the micro-computer system, various facilities of SCADA on RTU side are achieved. Data of analog and digital variables/quantities is acquired, integration of variables in the form of pulses is carried out and the data is processed and relevant information is displayed on CRT.

Relevant information is sent to MCS also. Provision is made to enable the operator to give control commands from keyboard.

The Master Control Station (MCS) is based on 80286 PC. It acquires relevant information from RTU at intervals. System configuration in the form of a mimic diagram is displayed along with the real time information. Hard copy of this display and data can be obtained on printer. The MCS issues control command for RTUs.

The communication of information between RTU and MCS is done via RS 232 C link using the standard three wire configuration. A protocol for exchange of information between the two stations, is designed in a manner as could ensure minimum error during communication.

The entire software for the RTU is written in the assembly langauge of 8085. The control and communication software for the MCS has been developed in the assembly langauge of 8086 while Fortran-77 has been used for writing the display software in the MCS.

Suggestions are made at the end for further work on the project.

CONTENTS

		Page
CANDIDATE'S DECLARATION		i
ACKNOWLEDGEMENT		ii
SYNOPSIS		iii-iv
CHAPTER-1	INTRODUCTION	1-1 - 1-9
1.1	What is SCADA	
1.1.1	Remote Terminal Equipment	
1.1.2	Communication System	
1.1.3	Computer System (Control Centre)	
1.1.4	SCADA Software	
1.1.4.1	Data acquisition software	
1.1.4.2	Supervisory control software	
1.1.4.3	Man-machine interface software	
1.2	Scope of Present Work-Objectives	
1.2.1	Facilities in RTU	
1.2.2	Facilities in MCS	
1.3	Organisation of Dissertation	
CHAPTER-2	LITERATURE SURVEY	2-1 - 2-13
2.1	Early Conception	
2.2	Later Manifestation	
2.3	Example for Telecontrol of a Remote Plant	
2.4	Trends in Communication	
2.4.1	Transistorized circuits	
2.4.2	Large automatic tracking antennas	
2.4.3	Phase locked FM Discriminators	
2.4.4	Communication Media	
2.5	Computers in SCADA	
2.5.1	Centralised computer control	
2.5.2	Distributed Computer Control	
2.5.3	Centralized or Decentralized	

contd.

Contents(contd.) Page

2.6	Modern SCADA Systems using Distributed Computer Control	
2.6.1	Necessity of Computer Control	
2.6.2	Heirarchical Control	
2.6.3	Description	
CHAPTER-3	REMOTE TERMINAL UNIT	3-1 - 3-19
3.1	Facilities Provided	
3.2	Hardware	
3.2.1	Hardware Description	
3.2.2	Use of the Data Acquired at RTU	
3.3	Software	
3.3.1	Data acquisition, Processing Software	
3.3.2	Application Example Software	
3.3.3	Display Software	
3.3.4	Communication Software	
3.3.5	Control Software	
CHAPTER-4	MASTER CONTROL STATION	4-1 - 4-4
4.1	Facilities Provided	
4.2	Hardware	
4.3	Software of MCS	
4.3.1	Communication Software	
4.3.2	Control Software	
4.3.3	Display Software	
CHAPTER-5	COMMUNICATION	5-1 - 5-12
5.1	Communication Link	
5.2	Communication Protocol	
5.3	Data Structure	
5.4	RTU Software for Communication	
5.5	MCS Software for Communication	

contd.

Contents(contd.) Page

CHAPTER-6 CONCLUSIONS AND SCOPE OF FURTHER WORK 6-1 - 6-3

6.1 Conclusions

6.2 Scope of Further Work

REFERENCES

APPENDICES:

A - REMOTE TERMINAL UNIT (RTU) SOFTWARE

B - MASTER CONTROL STATION (MCS) SOFTWARE

C - COMMUNICATION SOFTWARE

D - THE MODULAR MICRO COMPUTER SYSTEM TYPE EP-131

E - DOS INTERRUPTS

F - GRAPHIC SOFTWARE PACKAGE (GRAPH-X)

CHAPTER - 1
INTRODUCTION

1.1 WHAT IS SCADA

SCADA stands for supervisory control and data acquisition system. Supervisory control consists of telemetry and telecontrol. Telemetry implies measuring a quantity or quantities from a primary sensor, transmitting the results to a distant station, and then there interpreting, indicating and/or recording the quantities measured[1]. Remote control includes any system of control which requires a definite communication system to control action at a distance from the control point. The essence of the systems is that some part of the system must be located at a remote location. Interest in the remote location in most cases stems from the need to avoid a hostile environment, at the same time accomplishing the necessary measurement or control. Examples are aircraft test flights, missile remote control, nuclear reaction test, space-satellite monitoring, power plant monitoring, etc.

SCADA systems are typically found in industries where there are distant units of operation, often unattended, that require supervision from a control facility [2]. The main components of SCADA are :

1. Remote terminal equipment
2. Communication system
3. Computer system
4. SCADA software

1.1.1 Remote Terminal Equipment

The data is captured from the field by suitable transducers and special equipment and consolidated at remote stations in a microprocessor based terminal equipment (called as remote terminal unit or RTU)[2]. The RTU's at various stations send the data to the control centre under computer control via data communication links. Typically the RTUs perform the following functions:

1. Support communication line protocol and message formats
2. Maintain a local data base of the current state of field data.
3. Receive and analyse requests from the control centre
4. Execute control functions

The RTU works as telemetry and telecontrol equipment. It acquires, monitors and controls various parameters(generator voltage, bus voltage, generator power, CB condition etc.). It scans its inputs at predetermined intervals, compares the readings with previously stored data, thus enabling detection of any change of state and alarms[3]. This information is kept ready by RTU for onward transmission to the control centre when called for.

1.1.2 Communication system

It provides a path for data and control signals between RTU's and the computer system at the control centre[2]. Quite often existing communication links between stations and the control centre are used for data communication purposes in SCADA applications. The data communication media could be :

1. Voice-grade line
2. Power line carrier link
3. VHF link
4. Microwave/UHF link
5. Fibre optic link
6. Satellite link

A communication link may be a combination of one or more types with suitable interfaces. RTUs are connected via MODEMS to the communication systems. However, if the RTUs are in close proximity to the master control station, data is transmitted to the host computer in digital form using RS-232C links format and employing MODEMS or line drivers if necessary.

The security of the transmitted data distinguishes a SCADA system from normal data acquisition systems[3]. This security includes multiple data transmission, encryptions for coding the data, bit data techniques to assume correct messages and software for generating statistics on the number and types of errors.

Reliability of transmission and flexibility in communication are high priority items in SCADA systems[3]. The communication control modules that poll the RTUs and concentrate data for transmission to the host can handle more than one communication protocol to communicate with various systems.

Accuracy of data depends more on the sensors than the SCADA system. However, as accuracy demands increase, they will have an effect on transmission systems [3].

1.1.3 Computer System(Control Centre)

The computer at the control centre is responsible for gathering the data from the RTUs using pre-defined protocols. The following two types of protocols are used for acquisition of data [2].

1. Polling
2. Interrupt

In the polling protocol, the computer and RTUs follow the master/slave pattern where RTU sends data only on request from the master [2]. In the interrupt protocol the RTU sends data as soon as it is ready and this is treated as an interrupt by the control centre computer which processes it immediately.

Various configurations of the control centre computer are possible [2]. Earlier systems had a single computer to perform all functions. A modern trend has been to incorporate a front-end system to carry out the task of data acquisition.

Fig.1.1 shows a configuration overview of the computer system for a typical SCADA application [2]. The system reliability is enhanced by providing two super micro computers, the on-line system dedicated to the supervisory control of the remotes whereas off-line system used as cold standby or for the purpose of operator training [4]. It has been found necessary to provide such computers in order to maintain a reasonable scan time and yet provide for the large amount of data processing. Moreover

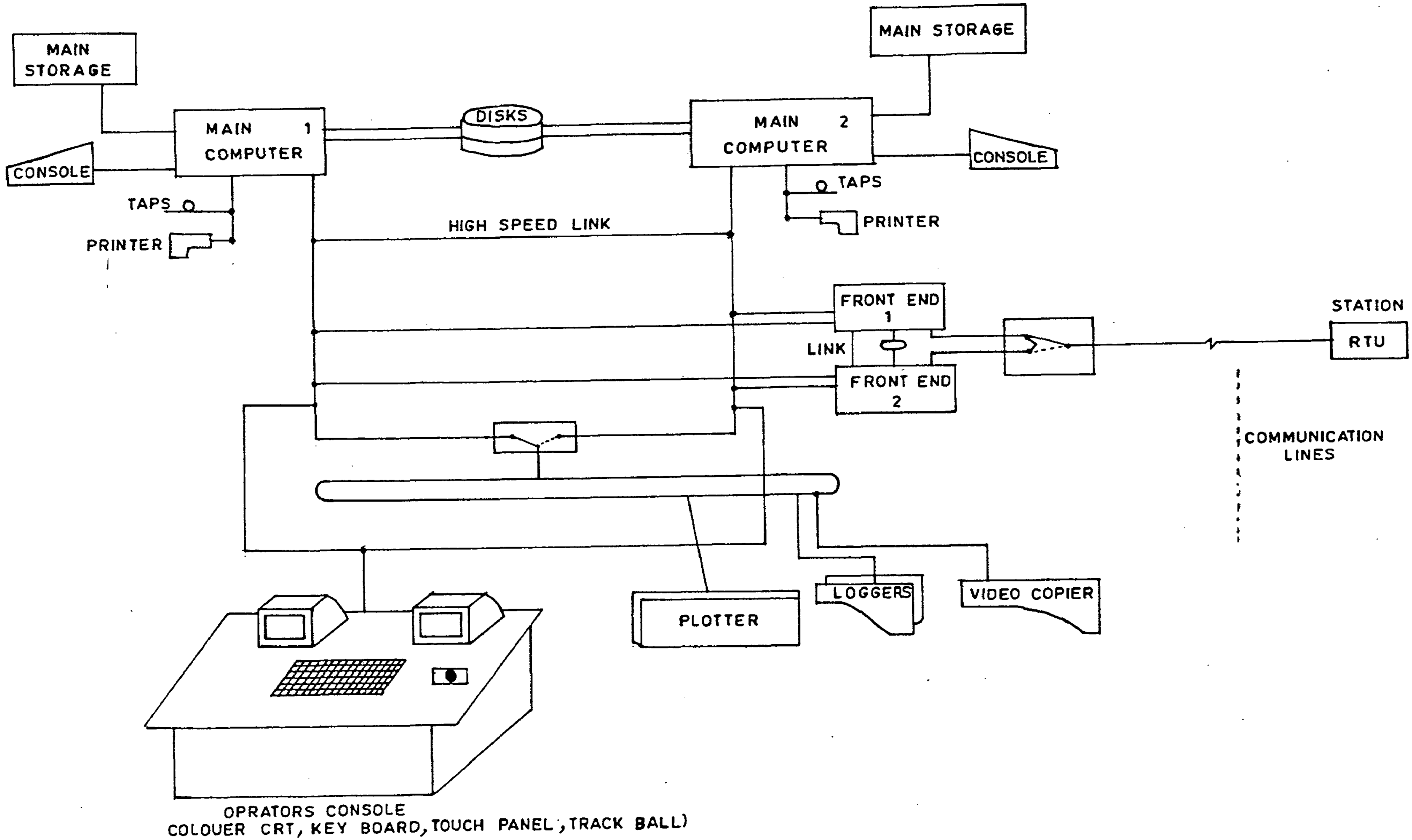


FIG. 1·1 COMPUTER SYSTEM AT A LARGE MCS.

the demand on processor time for task scheduling in a system such as this makes the choice of such large computers an unavoidable necessity.

The health of the on-line and off-line system is continuously monitored through Watchdogs and in case of on-line system failure a manual change over is done to standby system[4].

The Master Controller supports two operator workstations for controlling predefined and physically demarcated section of the OHE. Each work station is provided with two colour VDUs, one on-line and one standby and a keyboard containing functional and alphanumeric keys [4]. One of the VDUs can be configured as an operator console to view alarms, graphs etc. whereas the other can be configured as an engineer console from which commands can be given to the remote controllers. Various alarm, status, log and database displays and graphs, histograms, and trend curves are possible on master controller.

The Master Controller supports two printers which can be used to log alarm data and other either on demand or on event trigger [4].

A mimic diagramboard can be connected through suitable interface to the Master. This MDB provides a visual indication of the state of the equipment at the various stations.[4]. The dual front-end computers perform the functions of data acquisition[2]. The 'active' front-end carries out data acquisition from RTU and the other one (called 'standby') waits for the failure of 'active' computer to take over immediately.

1.1.4 SCADA Software

SCADA software can be classified as :

1. Data acquisition software
2. Supervisory control software
3. Man-machine interface software

1.1.4.1 Data acquisition software:

It supports data exchange between RTUs and computer systems at the control centre, generates the necessary commands for information required, performs error checking to ensure the validity of the data, proper completion of scan requests, and updates and maintains the data base [2]. It also provides support for the supervisory control functions by transmitting commands and performing error checks.

The data acquisition software allows for multiple cyclic scans(for obtaining data), each having assigned priority and interval between scans [2]. For each such scan, the software formats the appropriate data request, transmits the request and checks the return transmission for errors. All valid received data is then subjected to processing according to the data type. The received and any associated calculated data are then entered into the data base. Typical processing requirements are primarily oriented to detection and treatment of alarm conditions.

1.1.4.2. Supervisory control software:

This software is primarily responsible for the formatting of the control messages, transmission of the messages and validation of the check back responses according to the defined message protocol [2].

1.1.4.3 Man-machine interface software:

The software provides the following capabilities [2[

1. CRT Displays : Typically, the following picture types may be supported:

- Tabular data displays
- Trend - graphs of variables with time
- Bar charts
- Data entry forms
- Menus for selection of CRT pictures, data logs, and initiation of application programs.

2. Wall diagrams/strip chart recorders and other special analog and digital displays.

These are output devices only [1].

1.2 SCOPE OF PRESENT WORK - OBJECTIVES

Trends in SCADA system are studied. Some typical systems made by world class manufacturers are reviewed and compared. Important application of SCADA systems in India are also identified. On the basis of the state-of-art information " A MICRO COMPUTER BASED TWO LEVEL SCADA SYSTEM" of general purpose type has been developed. In this general purpose type system B0286- based PC is used for the master control station and 8085 based micro computer system for RTU (Remote Terminal Unit). Following facilities are provided in the MCS and RTU.

1.2.1 Facilities in RTU:

Data Acquisition:

Raw analog and digital data is acquired from the real world. Analog data is acquired with thehelp of an ADC unit and digital data with the help of digital I/O subsystem.

Data Display:

After acquiring and processing the analog and digital data it is displayed on CRT screen. The data is displayed in table format. Linking of CRT with 8085 is done serially with the RS232C link and all the communication between micro-computer & CRT is in serial form with a baud rate of 2400.

Control Functions:

A facility to control some analog variables by PID controller is provided. Some control commands can also be given by the operation through keyboard. Through one interrupt of 8259 (programmable interrupt controller) a start command is given for the plants. Through a second interrupt of 8259 a start command is given for the switching devices. One interrupt serves to switch off a failing switching device. This interrupt comes from the protection units of the switching devices.

1.2.2 Facilities in MCS:

The MCS has these following general purpose facilities:

1) It displays on PC monitor the information received from each RTU alongwith the mimic diagram of the controlled system. Data received from the various RTUs are displayed in sequence.

2) It gives all the control commands to the RTU including the reference values for the PID controllers.

3) It has a PID controller of its own.

Communication between RTU & MCS is serial via RS232C link using a USART (Universal synchronous & asynchronous receiver transmitter) at either end. Protocol is designed to ensure flawless transmission of data and repeating the message in case an error is detected by the receiver.

1.3 ORGANISATION OF DISSERTATION:

Followed by this introduction, a literature survey on the subject will be presented in the second chapter. The third chapter presents the details of the RTU, including the facilities provided, the hardware used, the software and the display of information on CRT. The fourth chapter discusses similarly the details of the MCS. Fifth chapter deals with the communication between RTU and MCS. Sixth and last chapter summarises the total work and brings forth the scope of further work.

CHAPTER-2
LITERATURE SURVEY

In this chapter, the trends in remote control, telemetry and supervisory control since the World War II are reviewed. Trends in communication, with respect to the impact of the development of transistor, large automatic tracking antennas, phase-I locked FM detector and technologically superior communication mediams, are also looked at. Finally the use of computers in SCADA is over-viewed. Some interesting and representative examples are also presented to highlight some important advancements.

2.1 EARLY CONCEPTION

An early conception of remote control was the bridge telegraph system between a ship bridge and engine room [5]. This system required human intervention to read signals and to activate the necessary control valves. Later, in the process plant control, these were actuated remotely. Telemetered data were used to establish the need for valve control and the extent of control.

2.2 LATER MANIFESTATION

The development of remote control has been principally centered around the drove and missile programs of the armed forces, that began during World War II[5]. Early droves were piloted air craft with the pilots removed and autopilots with remote radio control substituted. During test phases a pilot was usually carried to perform take offs and landings and to observe the results and deficiencies of the control equipment. Whenthe drove was used as a weapon the pilot was removed and the equipment functioned both automatically and by remote control. It soon became apparent that these missiles could be made smaller, of higher performance, and more economically if in their initial design no provision was made for a pilot.

One of the early missiles of pilotless design was designated as IB-2. Remote-control equipment from the drone programs was adapted to the IB-2 [6]. Telemetry was developed to measure the performance of the control equipment and the missile. In this evolution it is apparent that remote control equipment preceded telemetry by some years, but it was an "on-off" system rather than one permitting a continuous control. Furthermore, remote control was an intermittent function in as much as the vehicles were stabilized by internal automation equipment. Telemetry, on the other hand, required proportional and linear transmission of measurements on a continuous basis consequently, remote control systems were "adopted" in concept only and development of telemetering-equipment proceeded independently.

Another forerunner was telemetry and supervisory control in electric and gas utility transmission and distribution systems[6].

The public-utility measurements were made slowly, requiring only a very narrow band of frequencies for intelligence. With wire connections, there were no problems of radio fading, and many of the systems could be used only with continuous links between transmitter and receiver. Fades such as were normally occuring in radio systems would render the data valueless. Transducers were large and weighty, made for durability and easy servicing. They were not considered expendable and were chosen largely with a view toward long life and reliability; their response was slow.

Instead of techniques being borrowed from the utility field, the reverse trend has now appeared and utility telemetering has borrowed from the techniques of radio telemetering developed for missile testing.

2.3 EXAMPLE FOR TELECONTROL OF A REMOTE PLANT

A characteristic example for telecontrol of associated remote plant is the telecontrol of dam installations in hydro power stations which draw their driving water from a distant dam through a canal or a tunnel [7]. To use the water influx as effectively as possible, the control and supervision of the dam installation must be carried out from the power station control room. An early installation (year 1956) of great technical significance is the Runserau dam installation of the IMST power station in the Tyrol, Austria. The Runserau Dam installation of the Imst power station on the river Inn in the Tyrol was linked with the power station by a tunnel approximately 12.5 km long, which cuts off the bend of the river Inn at Landeek. The dam installation consisted of three sluice gate assemblies. It was manned by only one attendent who had the sole task of maintaining the mechanical equipment. The dam installation was remotely controlled from the power station.

However, since this could only be traversed during inspection periods, there would have been the fear that a possible breakdown in the cable could not be repaired for months or could require the emptying of the tunnel thus increasing the time of an operational failure [7]. The cost of the cable and its laying would have amounted to at least £ 10000, whereas renting costs for the 27 km long telephone link with two superimposed audio frequency channels amounted to about £ 3000 in ten years. The cost for the cable would thus be more than three times as large as this amount.

As the transmission method for both the commands for controlling the sluice gates, as well as for the signals from the dam installation, including the position values of the sluice gates and two water-level quantities, the pulse telegram method was chosen. The pulse telegram apparatus worked in both transmission directions each over an a.m. audio transmission channel on duplex traffic [7]. (Semi duplex equipment could not be used, so that the commands, especially the stop command, could be transmitted at any time even during the transmission of messages). The pulse telegram were transmitted in rest current operation over the audio transmission channels. An emergency stop command was superimposed on the command channel, if the command signal is not received for longer than about 0.5s in the dam installation, sluice gates which happen to be in motion are brought automatically to a halt. This ensures that the sluice gates, during a failure of the transmission line and audio transmission apparatus or after a breakdown in the pulse telegram equipment, cannot move further into an undesired position. (This is one of the most important safety requirements in the tele-control of dam installations). The telecontrol installation went into operation in 1956.

If it had been designed according to the state of telecontrol engineering at the time, the following major difference would be then:

1) FM audio transmission channels would probably be used.

2) In place of control by means of high, low and stop commands, servo control would be considered today.

2.4 TRENDS IN COMMUNICATION

The 50's saw the pulse telegram method as the transmission method for commands and messages [6]. The pulse telegram apparatus worked in both transmission directions each over an a.m. audio transmission channel on duplex traffic. The pulse telegrams were transmitted in rest current operation.

In the late 60's, electronic pulse methods with PCM or PDCM transmission were being used. FM audio transmission channels also came into use. Some major developments in the area of communication are reviewed below :

2.4.1 Transistorized Circuits:

The development of the transistor, and particularly the silicon transistor, has been very significant, especially to missile control [6]. It has permitted the reduction of size, weight, and power requirements - three factors which are of vital importance to missile operations. The replacement of the vacuum tube has been a gradual process, however, since stable operation over a wide range of temperature is more difficult with transistors. High frequency operation is just being achieved.

Other solid-state components were developed which can be used at micro-wave frequencies between 3 GHz to 10 GHz. These include the tunnel (Esaki) diode and the varactor. The former can be used as oscillator, switch, or r-f amplifiers and the later as switch or frequency multipliers. Microwave components based on travelling wave amplification have been developed with sufficient compactness and ruggedness for operation upto 10 GHz.

2.4.2 Large Automatic Tracking Antennas:

The development of large immovable parabolic reflecting antennas was largely accomplished under studies of forward-scatter propagation and the antennas were later adopted to telemetry use. [6]. The high gain of the large reflectors dictates that the beam width be relatively narrow and, therefore, tracking difficulties are presented in missile and satellite operations. It was not until the automatic tracking feature was added to the forward-scatter propogation antenna that it became practical for telemetering from guided missiles. Basically, there is a 10-db improvement in reception over previous techniques. This has made continuous data reception possible where otherwise there were losses due to fading. On the other hand, for the same performance characteristics, the missile transmitter power may be reduced by a factor of 10.

2.4.3 Phase-locked FM Discriminators:

Another improvement in telemetry receiving techniques has been the phase-locked frequency modulation (FM) discriminator, or detector[6]. The phase locked principle is one in which the frequency of a local oscillator is varied to correspond with the incoming frequency. This makes it possible to transmit the resulting beat frequency through a filter of narrow bandwidth. The local beat frequency oscillator is voltage controlled, the control voltage being the demodulated signal. The detector is a phase detector or multiplier instead of the conventional heterodyne detector. In a typical design, this technique added another 6-db of improvement to telemetry receiving stations, and this improvement increased to 15 db when the phase locked principle was also applied to the higher subcarriers [5].

2.4.4 Communication Media:

SCADA system's functioning is totally dependent on the communication system for transmitting voice, data and signal [8]. Power line carrier communication happens to be the most common form from earlier time and this is supplemented by switched telephone, radio communication and dedicated lines (Hot line despatcher-telephone system). Fibre optics is now becoming more popular communication medium due to technological superiority.

capabilities in the same mainframe. So these mainframe computer were superceded by minicomputers.

The mini computers were effectively superceded by the large centralised computer system for process control tasks in the early 1970's. They were developed to satisfy the requirements of DDC[9].

The large models of mini-computers, in fact, form the basis for nearly all supervisory control installations at present. However, for the smaller installations they have been superceded by the microprocessor.

2.5.2 Distributed Computer Control:

Withthe emergence of microprocessor and micro-computers an argument quickly developed in favourof distributed control.

As the control processes are often distributed over a wide area, it is natural that the computing power required to manage the plant is also distributed and concentrated where most work is required, to limit the data flow and achieve greater independence in case of failure of parts of the plant [9]. A general rule is that the structure of the control system should match the structure of the plant it controls. A process control system today is distributed among computing system whether or not it is portrayed in that way. Only the small data acquisition stations exhibit a concentrated structure.

2.5.3 Centralized or Decentralized:

A centralized system can respond faster, requires less interaction and the operator can control it better [9]. A decentralized system requires more local intelligence and imposes a communication overhead, but is less sensitive to partial outages and can be more easily tested and expanded. In fact the choice is quantitative, how much should be decentralized and how much to be centralized.

2.6 MODERN SCADA SYSTEMS USING DISTRIBUTED COMPUTER CONTROL

The advent of computerised process control and distributed control systems has enabled revolutionising the control of chemical and other industrious process, from simple parametric control to one of object and goal-oriented control system, leading to economic optimisation of the process. Petroleum refineries world-over have done pioneering work in fuller utilisation of the power of distributed control systems since the location of these refineries are geographically distributed. These systems mainly help in pushing the operating units to its maximum level of output, while keeping track of the dynamic constraints that exist in any given point of time.

2.6.1 Necessity of Computer Control:

The fundamental aim of process control is to keep a process value as close to a desired value, for as much time as possible. The proper solution of these desired process variable value depend on various factors such as through put, better product yields, consistent product qualities and demand of the products. While the conventional controls using either pneumatic or electronic instrumentation do a fairly good job in maintaining the individual process values to the desired value, they grossly fail to take care of the several interactions that exists between various variables and to that extent the control performance becomes inferior. Further there are unnumerable process values which are not directly measured, but are only calculated based on input of several variables. The conventional controls provide no answer to control them. Thirdly, the proper solution desired process values which is the main controlling factor is achieving the set objectives of the plant, are purely left to the operator's judgement. With the introduction of distributed control system with its supervisory computers answers to these problems have become possible. Computer control have come to state, in solving many of the vagging problems in the management and control of complex process industries and help to increase productivity.

The whole technology of computer control is built on the bottom-up control levels approach, which ensures certain level of controllability even when the higher level layer fails. This is very important aspect, without which the whole control system will fail like a pack of cards when failure occurs in one level.

2.6.2 Heirarchical control:

In large systems, the control is provided in four heirarchical levels. Functions incorporated at each level are as follows:

LEVEL-0:

Control of basic parameter using PID type regulating control.

LEVEL-1:

Dynamic computer control using certain advanced control techniques such as fuel forward control, adaptive gain control, calculation variable control etc.

LEVEL-2:

Optimisation of set points with minimisation/maximisation of suitably defined objective functionwhile ensuring quality and quantity of production.

LEVEL-3:

Time and space scheduling of production using techniques of operation research.

Fig.2.1 shows the general heirarchical structure of computer control.

2.6.3 Description:

LEVEL-0: aims at identifying basic regulatory controls like conventional flow, temperature, pressure and level controls. Proper pairing of variables are considered and basic loops are tuned scientifically based on process responses.

LEVEL 1 : is built over level-0, uses certain advanced control techniques. This is based on the stability and variability of the desired control. If interactions from other variables contribute substantially, it may be required to feed forward the effects of

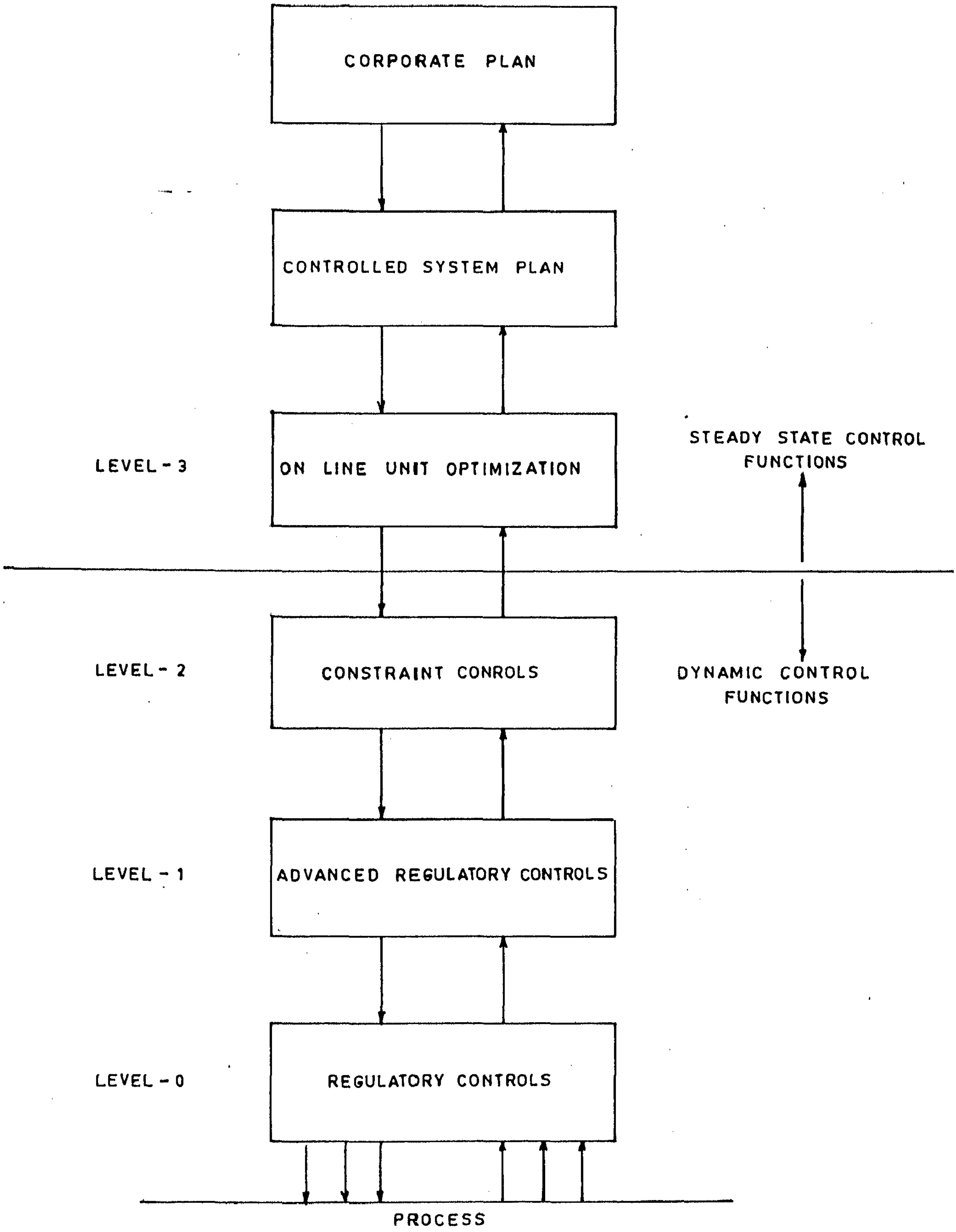


FIG. 2·1 CONTOL SYSTEM HIERACHY

these variables to control the desired variables. There may be instances where the desired variables to be controlled may not be a variable, which is directly measured. But it is controlled through other measured variables. In order to stabilise them, calculated or inferential controls are used. For example in a Distillation column, the desired controlled variable is the end point of the side draws. The variation in this affects the product quality. The variable which is controlled is the side draw product flows, which may not yield the desired stability in the end point. In such cases, advanced controls are used in which the calculated end points are the controlled variables and set point to the basic flow controllers are calculated to achieve the desired end point.

LEVEL-2 : constraint control is an important and easy computer control tool for optimisation of yield in many of the process applications. Thus constraint controls fit into LEVEL-2 in the hierarchy of computer controls.

In addition to constraint controls, LEVEL-2 controls also uses various optimisation techniques. This requires building up mathematical models of the problem and to derive optimum set points for critical variables using optimisation algorithms like linear programming, or generalised reduced gradient algorithm whenever non-linear optimisation is required.

LEVEL-3 : refers to problems of optimisation of whole process. Generally, optimisation should consider steady-state of the process rather than the dynamics which are handled by advanced control.

CHAPTER-3
REMOTE TERMINAL UNIT

This chapter discusses the details of the RTU of our system. Intel 8085 micro processor based card cage micro-computer system is used for hardware of RTU. With the help of different modules of the micro computer system, various facilities of SCADA on RTU side are achieved. All the facilities available in our system are given in detail along with an example for each facility. The hardware & software to achieve these facilities are described. Finally, detailed description on how the data or information is displayed on CRT is given.

3.1 FACILITIES PROVIDED

The following facilities are provided in the present system.

1. PID Control : PID control is used to control 4 analog variable.
2. 16-status inputs from switching devices
3. 16-status outputs for switching devices
4. 16-Analog inputs
5. Display : Out of 16 analog variable 4 analog variable are sampled at a rate of 1.25 ms and after the collection of every 16 samples their RMS values are calculated and displayed on CRT. Next 8 analog variables are sampled at a rate of 20 ms and after collection of every 50 samples, their average, minimum and maximum values are calculated. These values are displayed on the CRT

after every second. The last 4 analog variables are controlled by PID controller and there are also sampled at a rate of 20 ms. Reference and actual (instantaneous) values are displayed on the CRT repeatedly at an interval of 1 sec.

6. Monitoring : Maximum and minimum value of 8 analog variables are monitored and compared by their higher and lower limits stored in memory. In case of any variable exceeds its limit, one bit corresponding to the variable number is set in memory and an alarm is also given to indicate the fault.

7. Integration : Two variables are integrated by taking running sum of pulses representing the variable.

8. Control Command from Keyboard : Some of the operator control commands can be given by key board. The programs to achieve this is described under control software in software section.

9. Control Commands : Following two control command can be given by operator:

i) STOP SWITCHING DEVICE

ii) STOP PLANT

The functional block diagram of RTU is shown in Fig.3.1.(a). With the help of this system the above facilities are achieved.

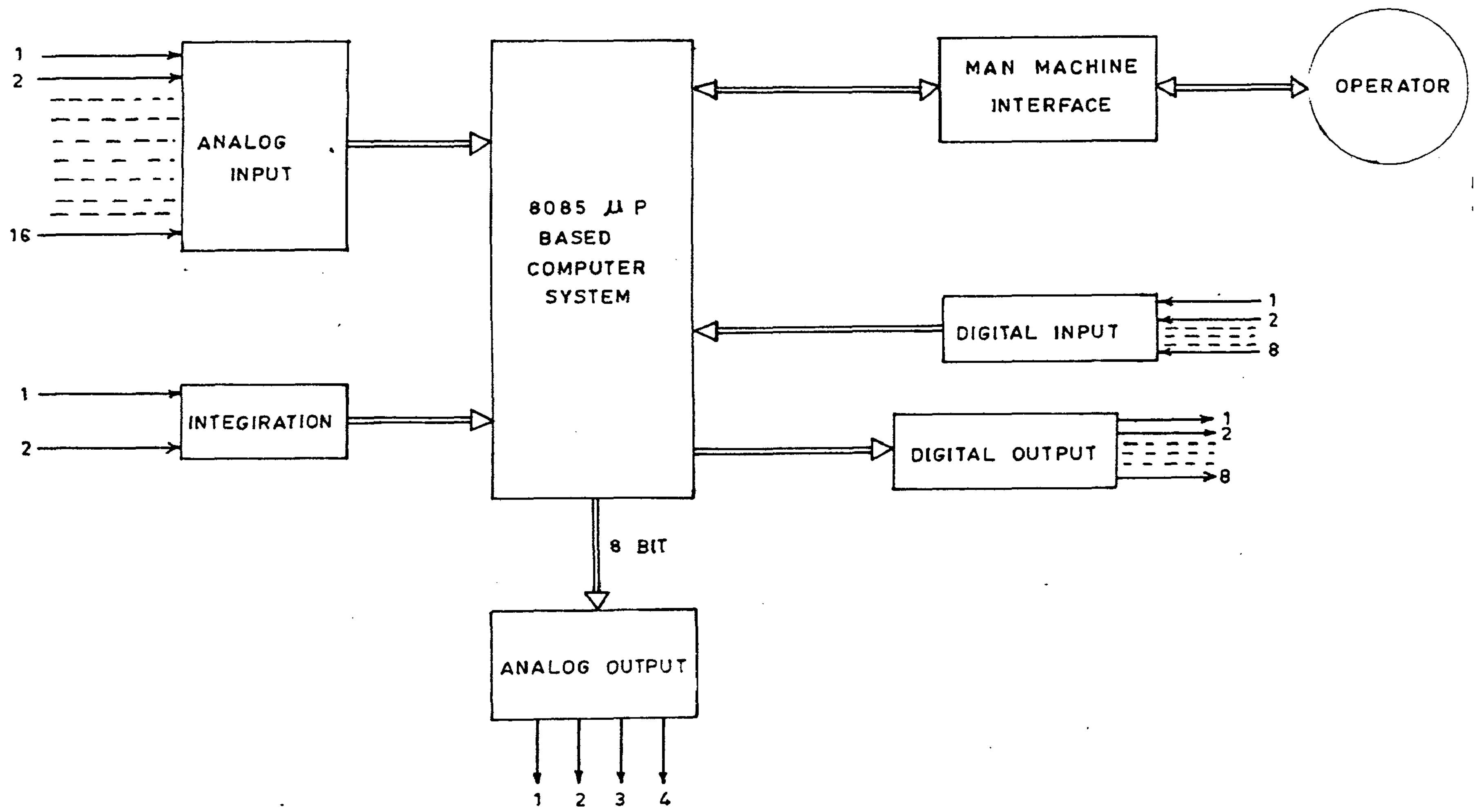


FIG. 3·1(a) FUNCTIONAL BLOCK DIAGRAM OF RTU

3.2 HARDWARE

3.2.1 Hardware Description:

The hardware of RTU is shown in Fig.3.1(b). Sixteen analog inputs are multiplexed through one Analog to digital convertors (ADC-573). Each ADC 573 is a 10-bit ADC.4 of these inputs are controlled by a software PID controller and the controlled outputs are fed to digital to analog controller (DAC 0800). 16 digital inputs are fed through 8255 (programmable peripheral interface) to the 8085 µp. To control 8 switching devices, control commands are given via a port of 8255. Similarly the control 8 plants 8 control commands are given by 8255.

A programmable timer of IC 8253 is made to generate a periodic interrupts at every 1.25 msec. Two counters of IC 8253 are used to count the pulses from integral device.

4 control commands can be given via the 4 interrupts of 8259 (Programmable interrupt controller). Each interrupt of 8259 is a logical OR of 8 interrupts from different devices. To identify the interrupting device, the ports of 8255 are polled. The processed data is displayed on CRT, which is interfaced serially to 8085 µp through USART (Universal synchronous asynchronous receiver transmitter IC 8251). The communication between the RTU up and the Master Control Station (MCS) also takes place via 8251.

The keyboard is interfaced to the µp via IC 8279 (key board interrupt controller).

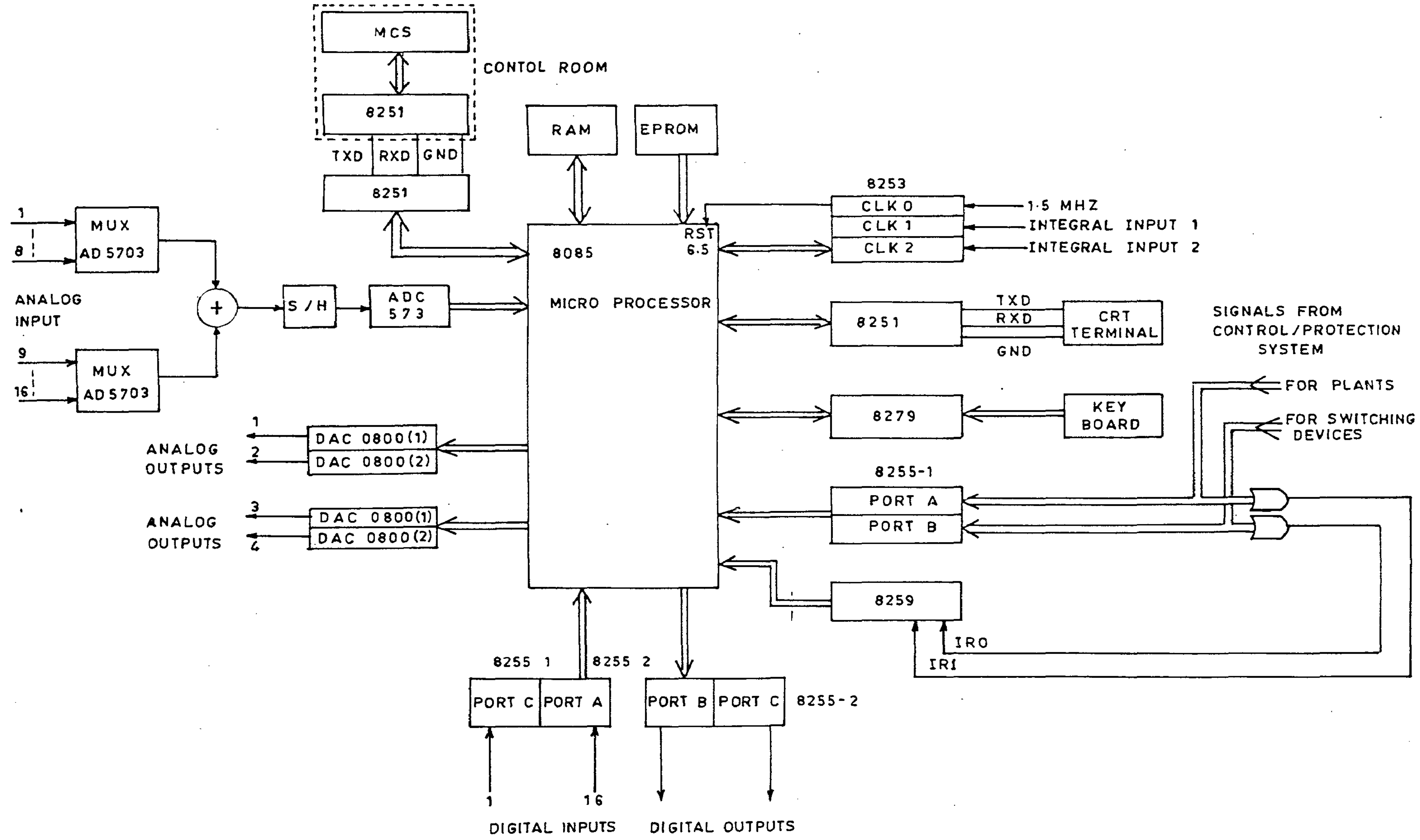
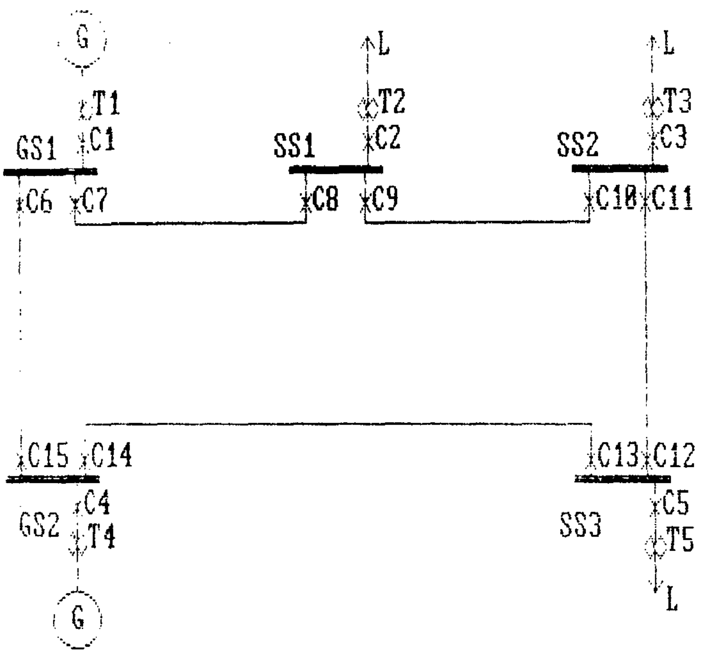


FIG. 3·1 (b) HARDWARE SCHEMATIC OF R.T.U



CIRCUIT BREAKER STATUS

CB1	ON	CB9	OFF
CB2	ON	CB10	OFF
CB3	ON	CB11	ON
CB4	ON	CB12	ON
CB5	ON	CB13	ON
CB6	ON	CB14	ON
CB7	ON	CB15	ON
CB8	ON		

BUS VOLTAGE(KV)

GS1	132
GS2	132
SS1	132
SS2	132
SS3	132

	VOLTAGE(KV)		POWER(MW)		MAX TEMP(C)	
	REF	INSTA	REF	INSTA	BEARING	WINDING
GEN1	11	10.8	100	101	80	60
GEN2	15	10.2	200	198	85	65

DIS. VOLTAGE(KV)

SS1	33
SS2	11
SS3	11

Temporary storage of program and intermediate data are stored in a RAM area of addresses B000- BFFFH. RAM address area is also between 4000H to AFFF C000 to DFFFH. IC 6264 (RAM) chips are used and IC 2716 (ROM) chips are used in RAM and ROM cards respectively.

3.2.2 Use of the Data Acquired at RTU :

We have assumed that our RTU is for the (generating station-1) (GS 1) of Fig.3.2. The Fig.3.2 is example of a power system generating and distributing network.

Following variables are sensed at this generating station.

a) Generator Voltage:

Generator is generating power at a certain voltage and that voltage should remain constant at a value, so that voltage is sensed and controlled by PID controller. The reference of the voltage is set by Master Station or by operator.

b) Generator Power :

How much power is generated by generator is sensed and it is again a controlled variable. The generator power is again controlled by a PID controller whose reference value is given by Master station or operator.

The two controlled variables are sampled at a rate of 20 ms.

c) Bus voltage:

The bus voltage after the transformation of the generator voltage is sensed. This parameter is a uncontrolled parameters which is sampled at a rate of 1.25 ms and after one cycle its rms and average values are calculated. The rms value of this variable is sensed and given to master station.

d) Bearing Temperature :

The temperature of bearing is sensed at 0 rate of 20 ms and after every 1 sec i.e. after collection of 50 samples its maximum temperature is seen and sent to master station. The maximum temperature is also checked up by its higher limit. If it is exceeding its higher limit operator can give a command to stop the plant.

e) Winding Temperature :

The temperature of winding is sensed at a rate of 20 ms and after every 1 sec i.e. after collection of 50 samples its maximum temperature is also checked up by its higher limit. If it is exceeding its higher limit operator can give a command to stop the plant.

f) Positions of CB1, CB6, CB7 are sensed. These inputs are the digital input in form of 3 bits. If bit is set implies that CB is closed (ON) else OFF.

g) Energy generated by generator is also sensed by pulsed input and that can be read by master by giving command.

3.3 SOFTWARE

Software for RTU is divided into several subsection according to the use of software.

3.3.1 Data Acquisition, Processing Software:

a) Input Analog Value:

This routine inputs the analog variable for the specified channel. In this routine channel number is latched and then a delay of 20 μs is called, so that conversion (Analog to Digital) is over by Analog to Digital Convertor (ADC). Then higher and lower bytes of digital O/P is read. The lower byte contains useful information only in two L.S.Bs, so the higher 6 bits are masked. The lower 6 bits of higher byte are shifted to the higher 6 bits of lower byte and bit 6 and 7 of higher byte are shifted to bit 0 and bit 1. So the 10 bit of result are placed properly in the memory. Flow chart is shown in Fig.3.3.

b) Delay :

This routine gives a delay of 20 μs. The delay count is in register pair BC and the count is decremented till it is zero which gives a delay of 20 μs. The flow chart is shown in Fig.3.4.

c) Output Analog Value :

This routine outputs the equivalent analog value of 8-bit data. In this routine initialise the channel number and output

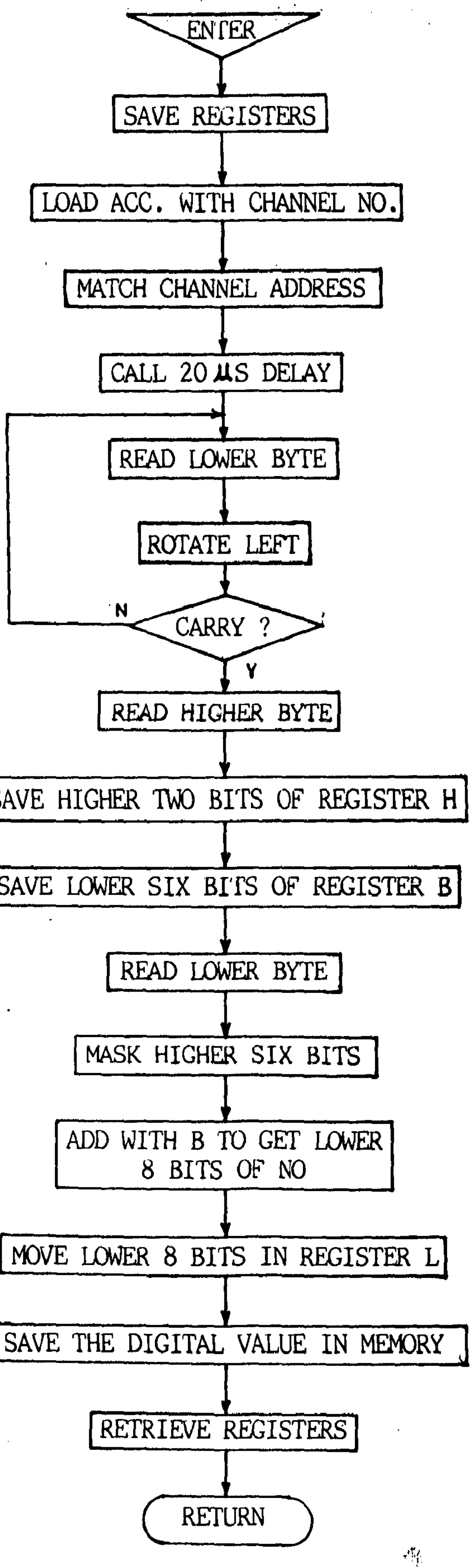


Fig.3.3 : Routine to Read Analog Inputs

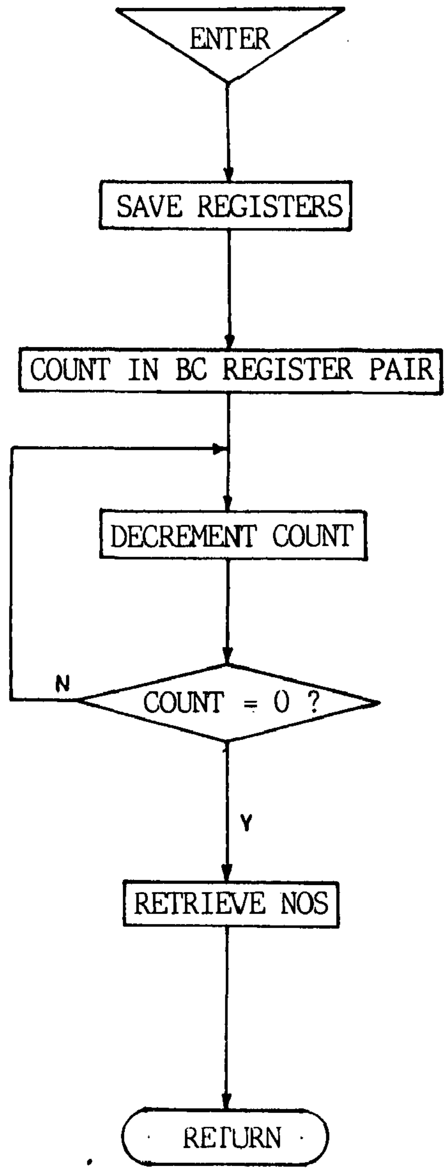


Fig.3.4 : Routine to give 20 μs delay

the digital value to that channel. The flow chart is shown in Fig.3.5.

d) Input digital value :

This routine inputs the 16-bit digital input via input port of 8255. The digital values are the status of 8-switching device and 8 plants.

e) Output digital value :

This routine outputs the control command (8-bit) to 8-switching devices. and for 8 plants also.

f) Multiplication (16-bit by 16-bit multiplication):

This subroutine multiplies the contents of Register pair D by the contents of register pair B. Multiplier and multiplicant from the memory are load in subsequent register pairs. The 32-bit result will be contained in register pair H (the two least significant bytes) and register pair D (the most significant bytes). Finally the result is stored in memory. The flow chart for this routine is shown in Fig.3.6.

g) Division (16-bit by 16-bit division):

This subroutines divides the 16-bit quantity in register pair D by the 16-bit quantity in register pair B. The divident in DE and divisor in BC can be loaded from memory. The result can be stored in register pair D, which is finally stored in a memory location. The flow chart for division routine is shown in Fig.3.7.

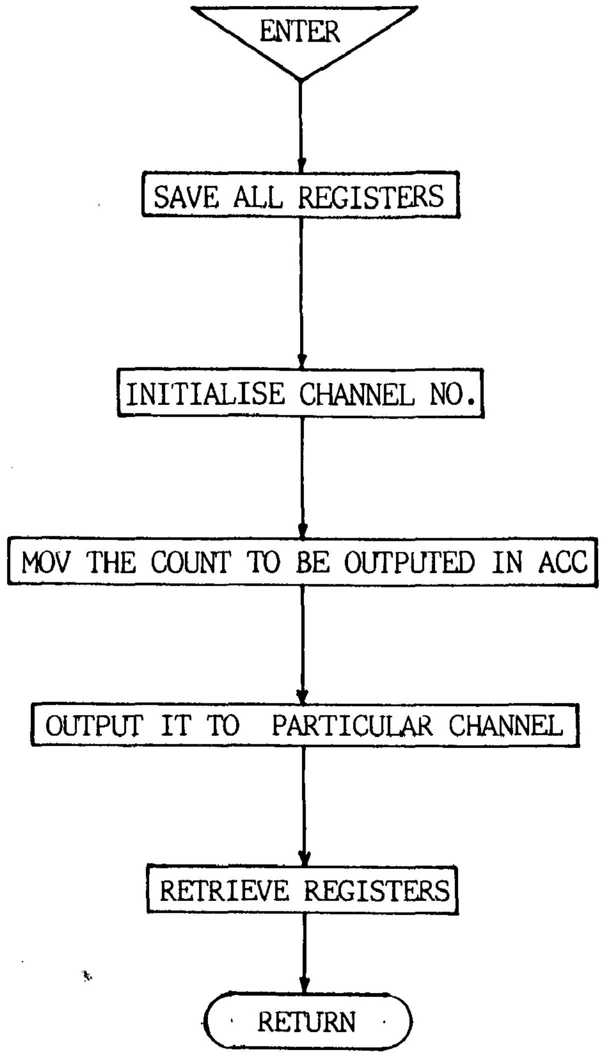
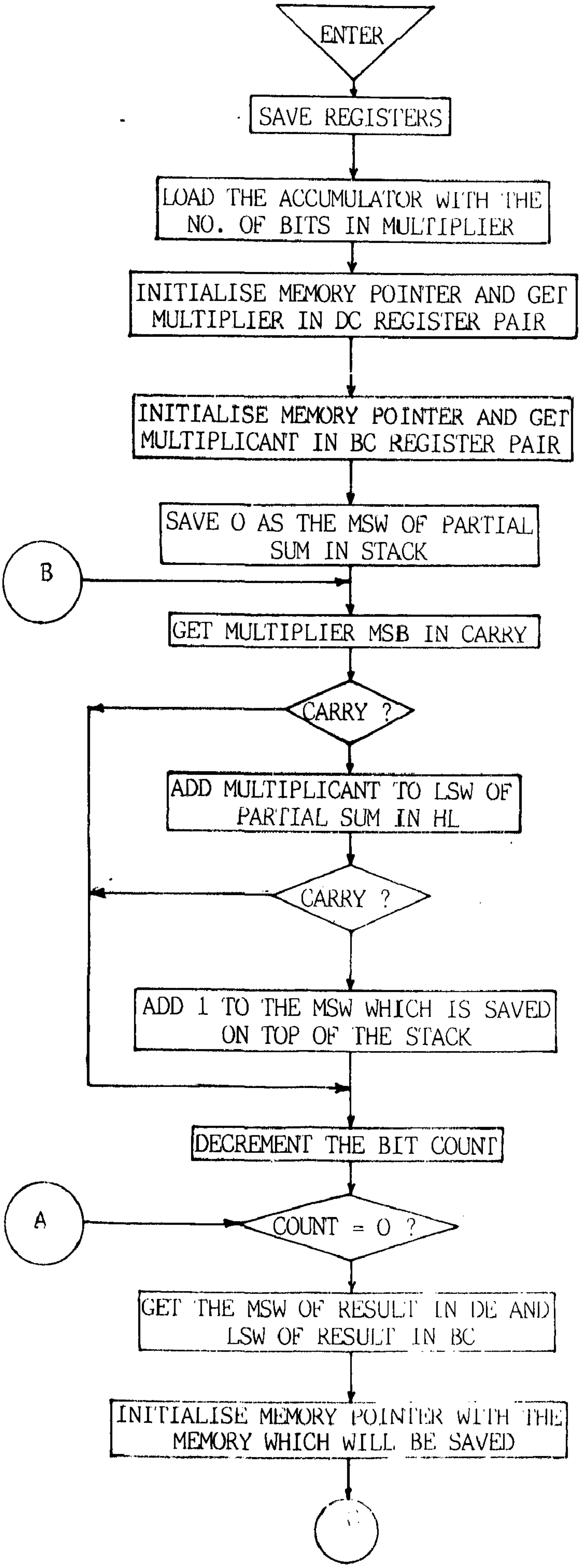


Fig.3.5 : Routine to output analog value


ENTER
SAVE REGISTERS
LOAD THE ACCUMULATOR WITH THE NO. OF BITS IN MULTIPLIER
INITIALISE MEMORY POINTER AND GET MULTIPLIER IN DC REGISTER PAIR
INITIALISE MEMORY POINTER AND GET MULTIPLICANT IN BC REGISTER PAIR
SAVE 0 AS THE MSW OF PARTIAL SUM IN STACK
B
GET MULTIPLIER MSB IN CARRY
CARRY ?
ADD MULTIPLICANT TO LSW OF PARTIAL SUM IN HL
CARRY ?
ADD 1 TO THE MSW WHICH IS SAVED ON TOP OF THE STACK
DECREMENT THE BIT COUNT
A
COUNT = 0 ?
GET THE MSW OF RESULT IN DE AND LSW OF RESULT IN BC
INITIALISE MEMORY POINTER WITH THE MEMORY WHICH WILL BE SAVED

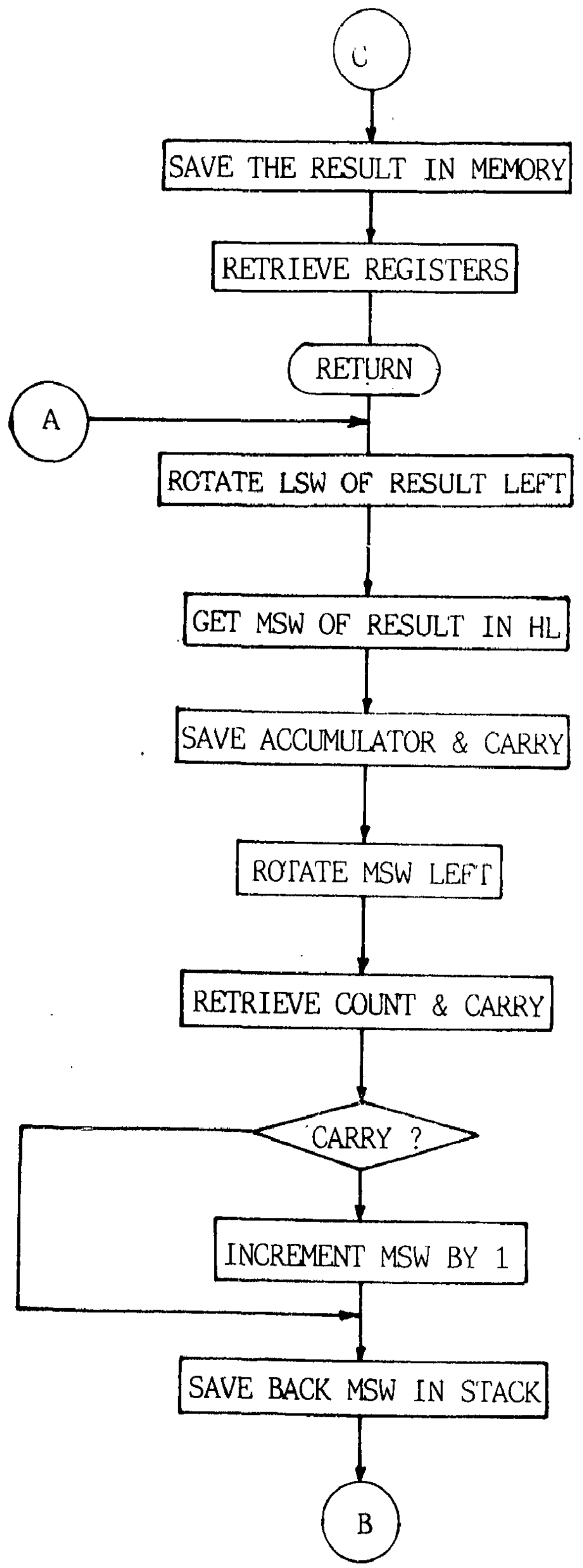
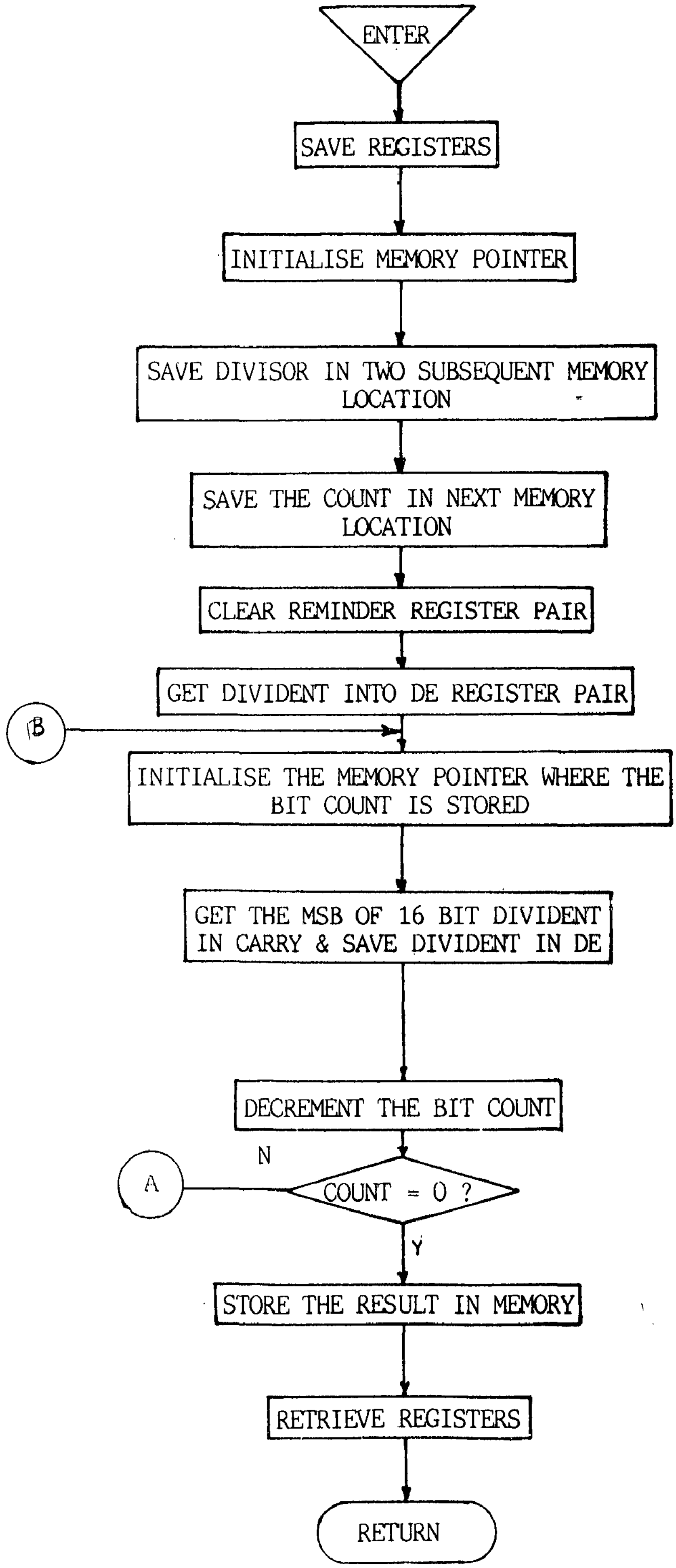


Fig.3.6: Routine for 16 x 16 Bit multiplication


ENTER
SAVE REGISTERS
INITIALISE MEMORY POINTER
SAVE DIVISOR IN TWO SUBSEQUENT MEMORY LOCATION
SAVE THE COUNT IN NEXT MEMORY LOCATION
CLEAR REMINDER REGISTER PAIR
GET DIVIDENT INTO DE REGISTER PAIR
B
INITIALISE THE MEMORY POINTER WHERE THE BIT COUNT IS STORED
GET THE MSB OF 16 BIT DIVIDENT IN CARRY & SAVE DIVIDENT IN DE
DECREMENT THE BIT COUNT
N
A
COUNT = 0 ?
Y
STORE THE RESULT IN MEMORY
RETRIEVE REGISTERS
RETURN

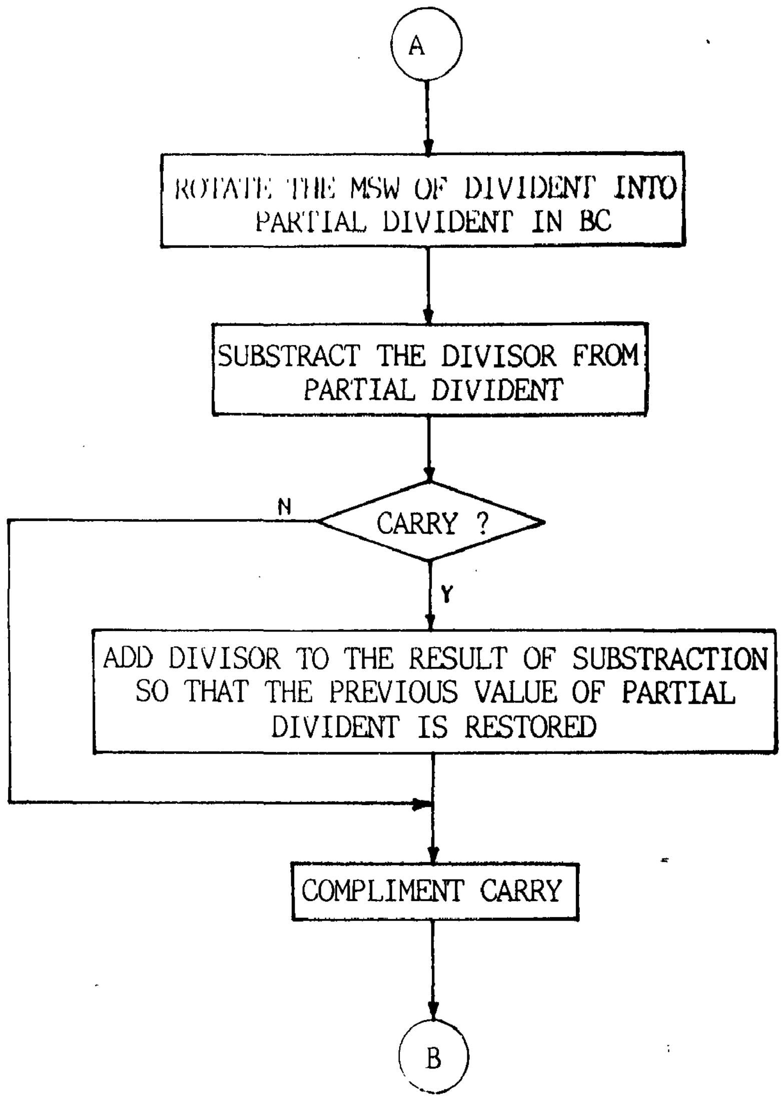


Fig. 3.7 : Routine for 16 x 16 Bit Division

h) Addition (32-bit):

This subroutine adds the contents of four consecutive memory location to the content of register B,C,D and E. Register E contains the least significant byte,and stores in memory location with the lowest memory address. The result is finally stored in memory. Largest number for this system is 20 bit number and maximum 50, 20 bits no. canbe added, so the result never exceeds by 32 bit. The flow chart is shown in Fig.3.8.

i) Average (16-Bit):

This subroutine calculates the average value of 50 or 16, 16-bit data stored in sequential memory location. The ADC O/P is a 10 bit output. The result of addition of 50, 10-bit (max) number will never be more than 16-bit. So the procedure of 16-bit addition is right. After getting the addition result the result is then divided by the count and finally the average value is calculated and stored in memory. The flow chart is shown in Fig.3.9.

j) Binary to ASCII conversion :

Binary data is converted to ASCII before being output to display device. Binary numbers are easily converted to BCD through repeated division by binary ten. Then 30H is added to this BCD no. to convert it into ASCII No. This method is useful if the microprocesses has a divide instructioin. A binary to ASCII conversion method that is useful when a divide instruction is not

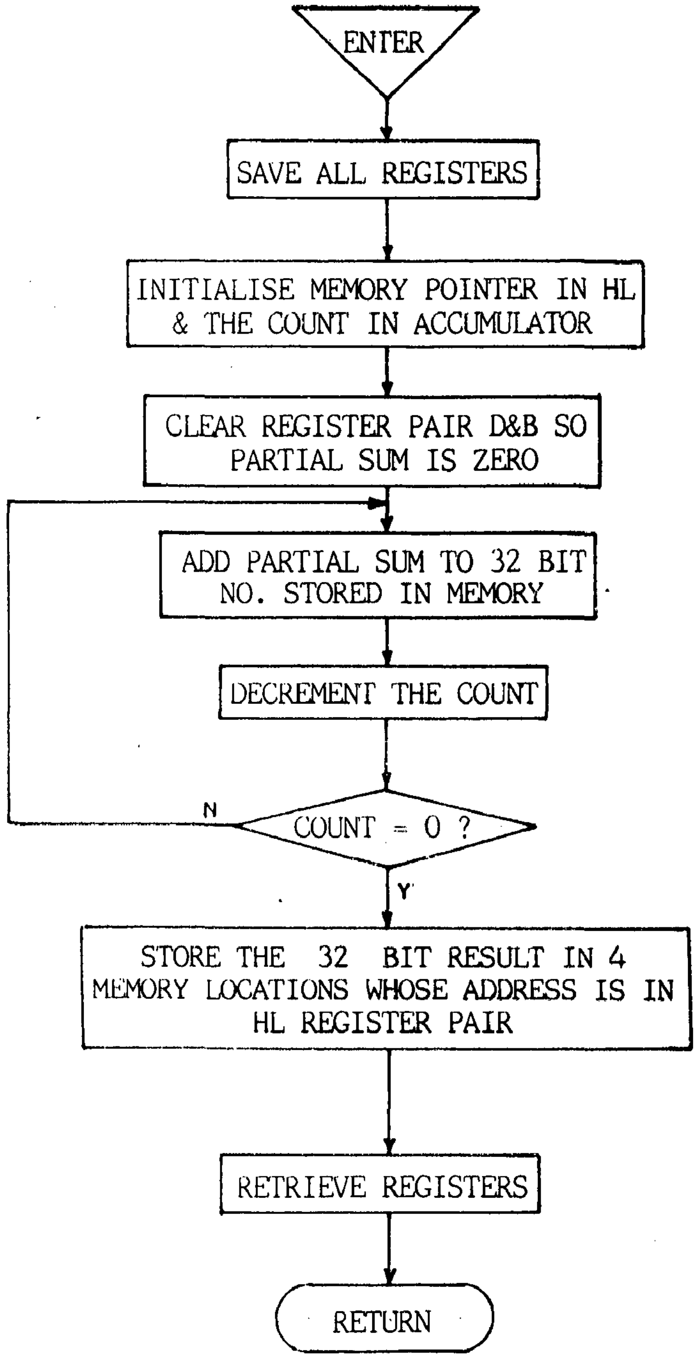
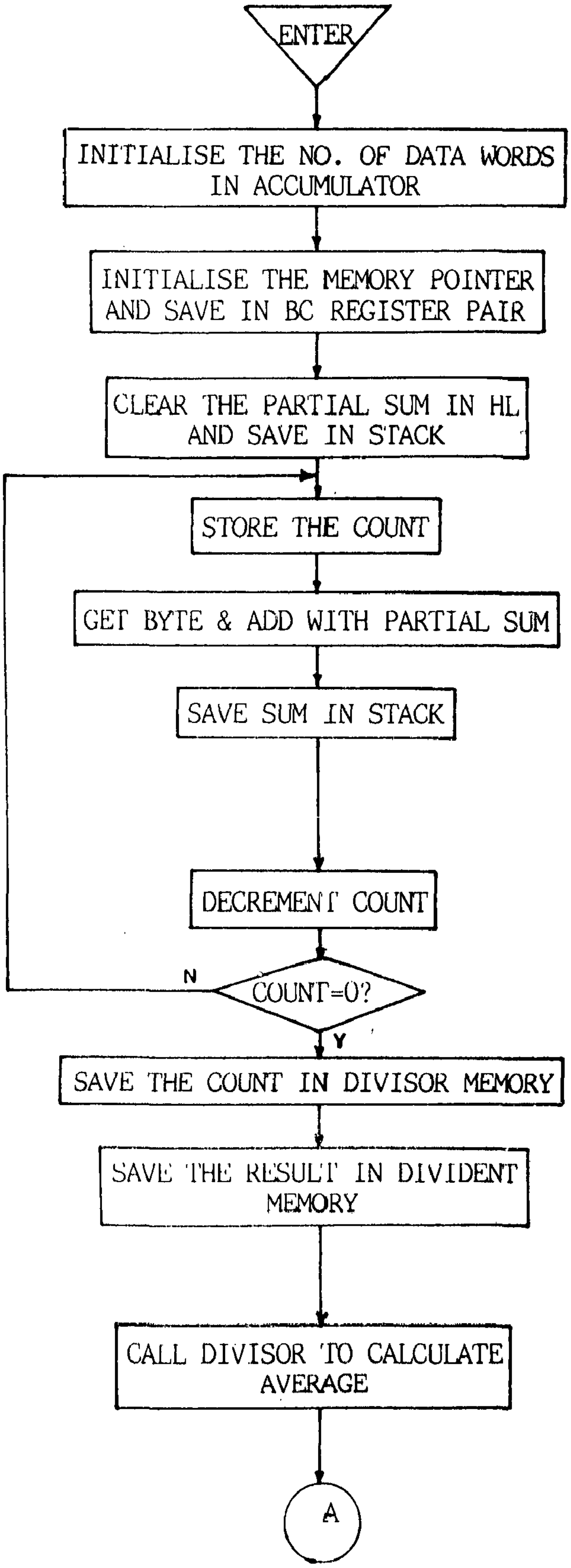


Fig.3.8 : Routine for 32 Bit Addition


ENTER
INITIALISE THE NO. OF DATA WORDS IN ACCUMULATOR
INITIALISE THE MEMORY POINTER AND SAVE IN BC REGISTER PAIR
CLEAR THE PARTIAL SUM IN HL AND SAVE IN STACK
STORE THE COUNT
GET BYTE & ADD WITH PARTIAL SUM
SAVE SUM IN STACK
DECREMENT COUNT
COUNT=0?
N
Y
SAVE THE COUNT IN DIVISOR MEMORY
SAVE THE RESULT IN DIVIDENT MEMORY
CALL DIVISOR TO CALCULATE AVERAGE
A

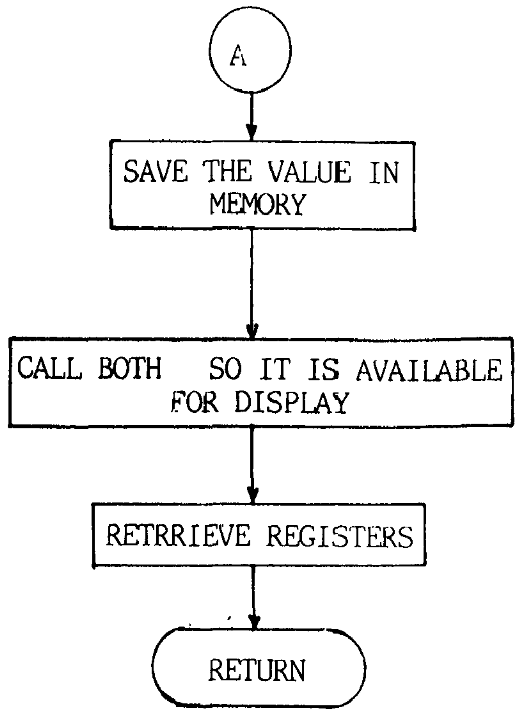


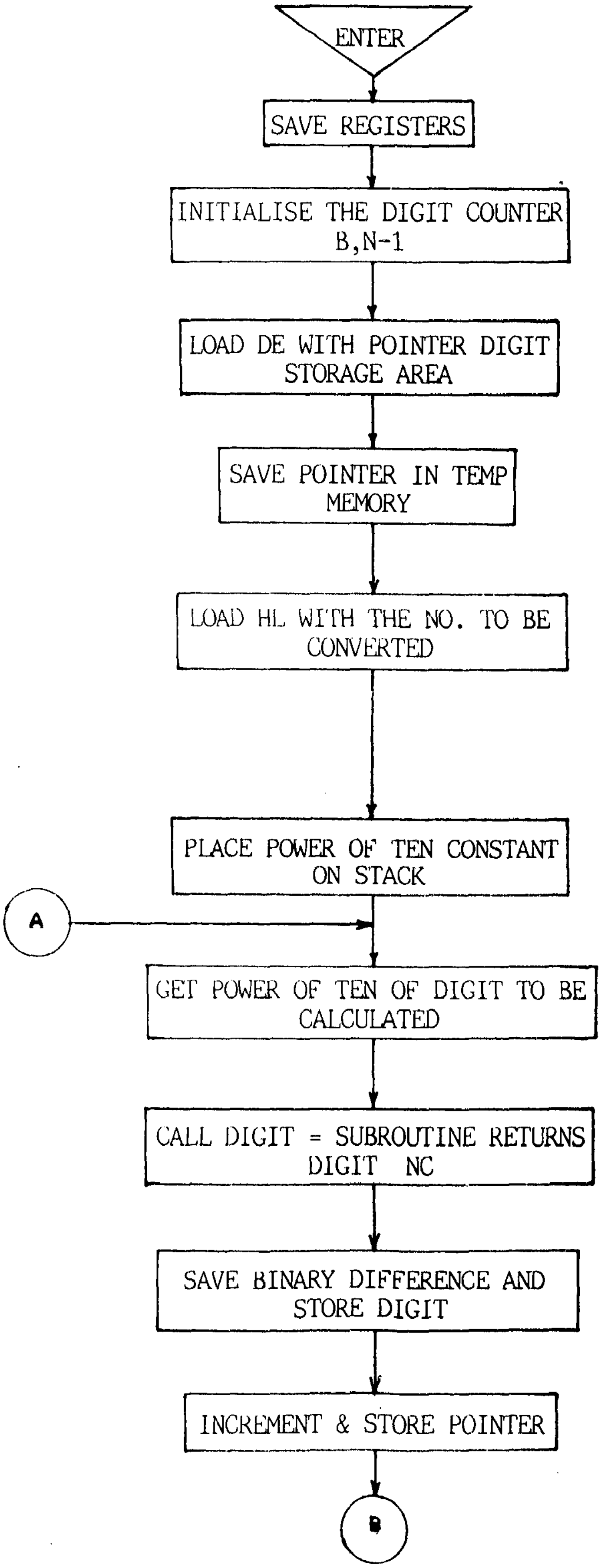
Fig.3.9 : Routine to calculate average of N 16 Bit Numbers

available is repeated substraction if powers of ten in binary. The highest power of ten possible in the binary number is repeatedly subtracted from the no. until the difference becomes negative. The number of times the subtraction can be accompolished without a negative difference provides the digit associated with the power of ten being subtracted. The next highest power of ten is then substracted from the positive binary difference resulting from the determination of previous digit. When the digit associated with 10° is obtained, the positive remainderis the digit corresponding to 10°. Every time a decimal digit is obtained add 30H to convert it into ASCII. This procedure (shown in Fig.3.10) converts the binary no. in HL pair to its ASCII equivalent. Binary no. can be laod in HL from memory and finally the ASCII no. is also stored in memory.

k) Square root (32-bit number):

Simplest way of finding the square root of a number is by using successive approximation algorithm. Successive approximation works as follows :

Let B be the value for which the square root is desired and A be the guess value of B. The value of A is squared and compared to the value of B. If A^2 is greater than B, A is decreased, but if A^2 is less that B, then A is increased by any arbitary number. This procedure is repeated until A^2 is approximately equal to B. The user decides how close A^2 needs tobe compared to B before the loop is terminated.


ENTER
SAVE REGISTERS
INITIALISE THE DIGIT COUNTER
B,N-1
LOAD DE WITH POINTER DIGIT
STORAGE AREA
SAVE POINTER IN TEMP
MEMORY
LOAD HL WITH THE NO. TO BE
CONVERTED
PLACE POWER OF TEN CONSTANT
ON STACK
A
GET POWER OF TEN OF DIGIT TO BE
CALCULATED
CALL DIGIT = SUBROUTINE RETURNS
DIGIT NC
SAVE BINARY DIFFERENCE AND
STORE DIGIT
INCREMENT & STORE POINTER
B

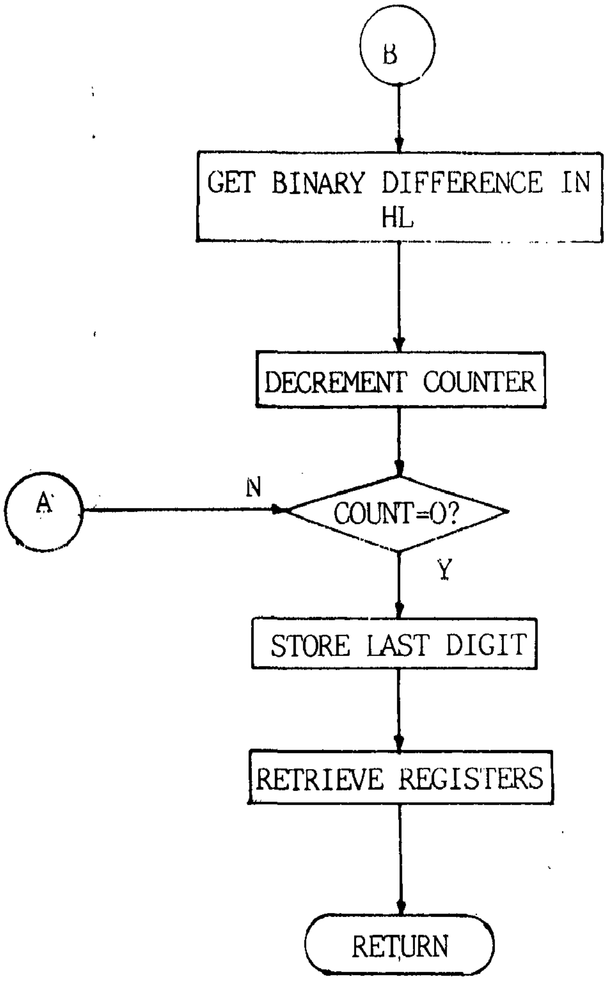


Fig.3.10.1 Part of Binary to ASCII Conversion

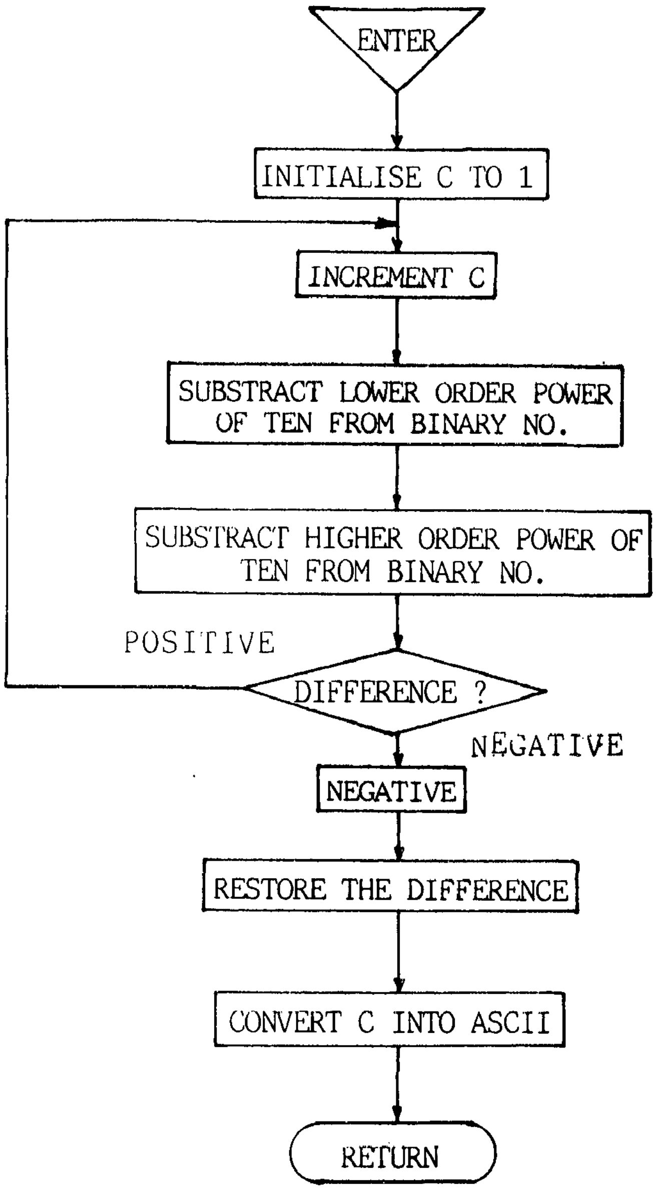


Fig.3.10.2 : Digit Calculation

Fig.3.10 : Routine for Binary to ASCII Conversion

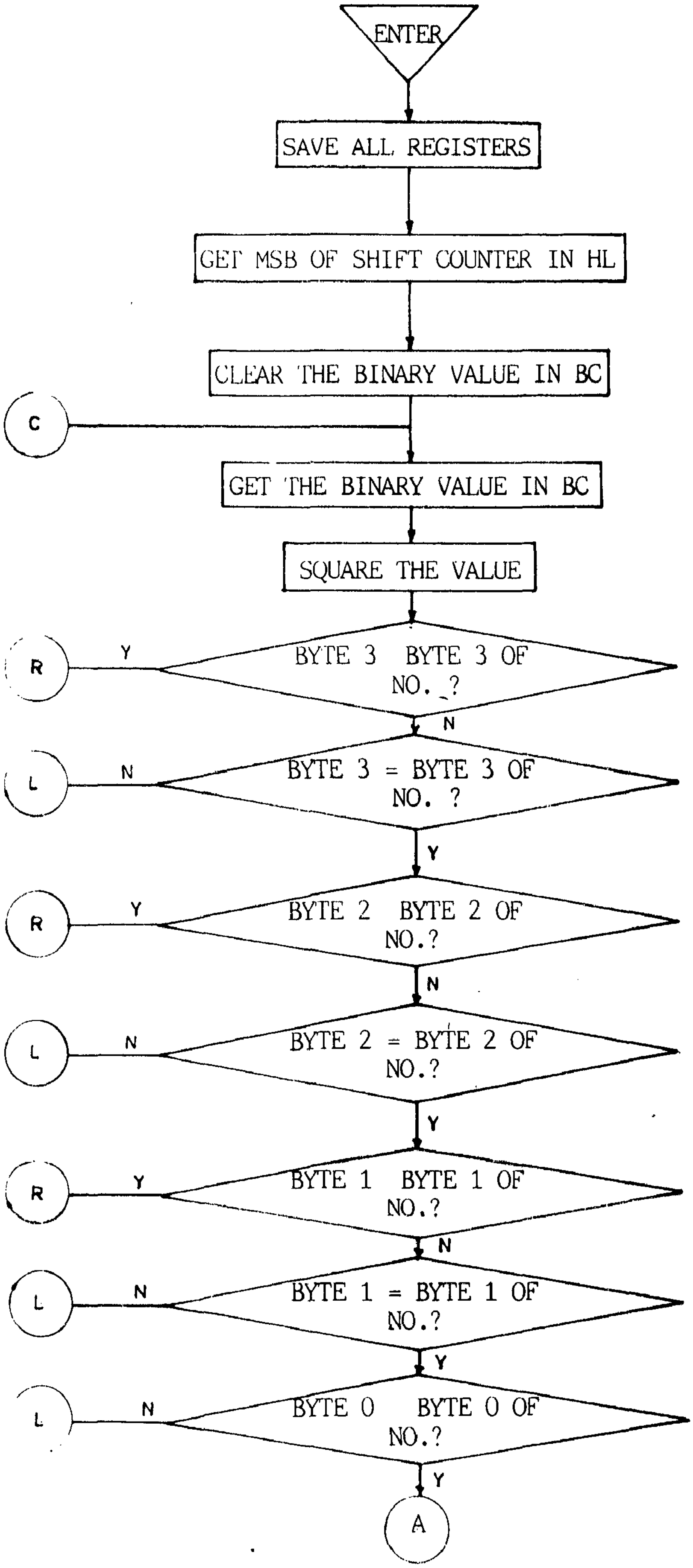
Calculation of square root of a 32-bit no. is shown in Fig. 3.11. The no. whose square root has to be calculated is stored in memory location MEM100, MEM101, MEM102, MEM103. Highest byte in memory location MEM103 and lowest byte in MEM100. The approximate no. A is taken as the middle of the 16-bit No. i.e. the comparison starts from 8000H. And If $A^2 > B$ than highest bit is reset and next higher bit is set i.e. 4000H. And If $A^2 < B$ than the next higher bit is also set COOO H. Like this way the comparison is performed. The square root of 32 bit no.by the comparison is performed. The square root of 32 bit no. is a 16-bit no. in register pair BC which is finally stored in memory.

l) Square (This program square the no. in BC register pair):

This routine squares the 16 bit no. in register pair BC. As for this routine multiplier and multiplicand are the same so load the contents of BC pair to the DE pair also. The result will be stored in memory location (MEM107 to MEM 104). The highest significant byte is MEM107 and lowest significant byte of result in MEM 104. The flow chart is shown in Fig.3.12.

m) RMS (16-bit numbers):

This routine calculates the RMS of sixteen 16-bit numbers stored in sequential memory location. The logic used for RMS calculation is first the squares of all sixteen nos. are calculated and again the result is stored in sequential memory location. Then addition of these sixteen nos. by calling ADD32 subroutine is done. Now instead of calculating the mean of the result first


ENTER
SAVE ALL REGISTERS
GET MSB OF SHIFT COUNTER IN HL
CLEAR THE BINARY VALUE IN BC
C
GET THE BINARY VALUE IN BC
SQUARE THE VALUE
R
Y
BYTE 3 BYTE 3 OF NO. ?
N
L
N
BYTE 3 = BYTE 3 OF NO. ?
Y
R
Y
BYTE 2 BYTE 2 OF NO.?
N
L
N
BYTE 2 = BYTE 2 OF NO.?
Y
R
Y
BYTE 1 BYTE 1 OF NO.?
N
L
N
BYTE 1 = BYTE 1 OF NO.?
Y
L
N
BYTE 0 BYTE 0 OF NO.?
Y
A

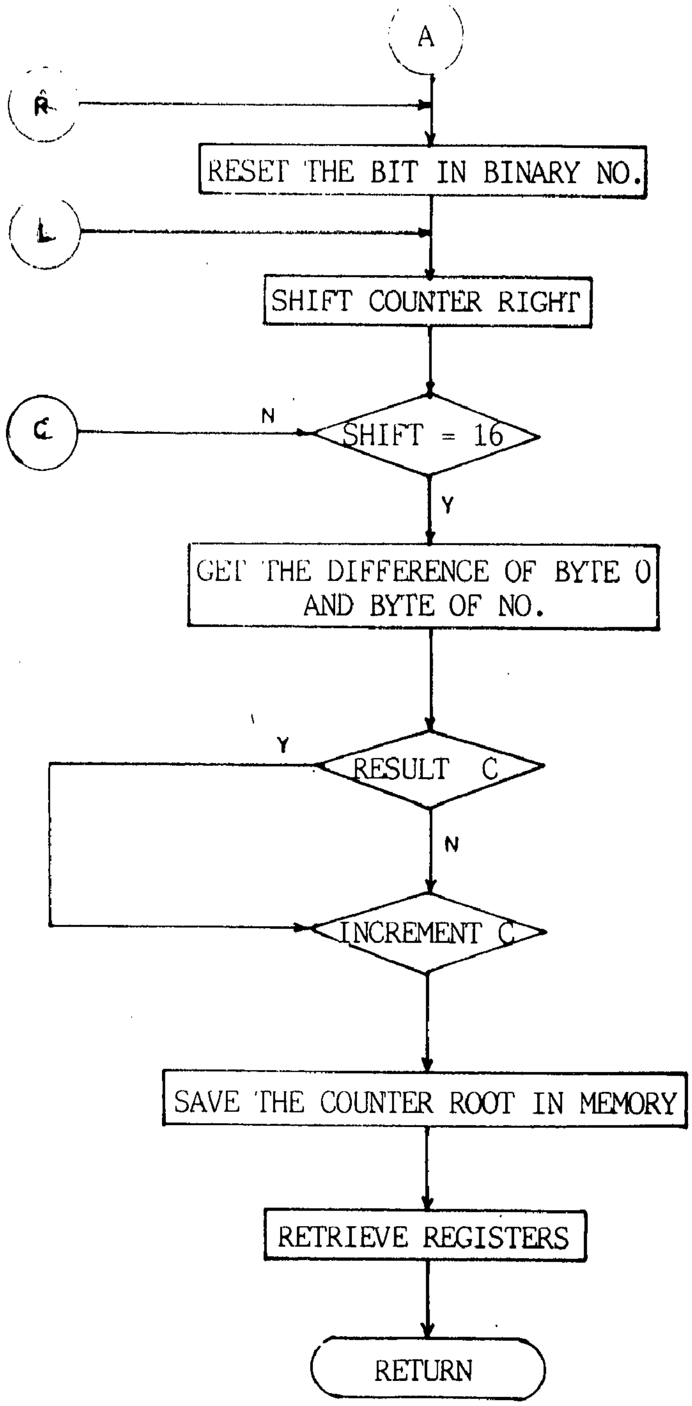
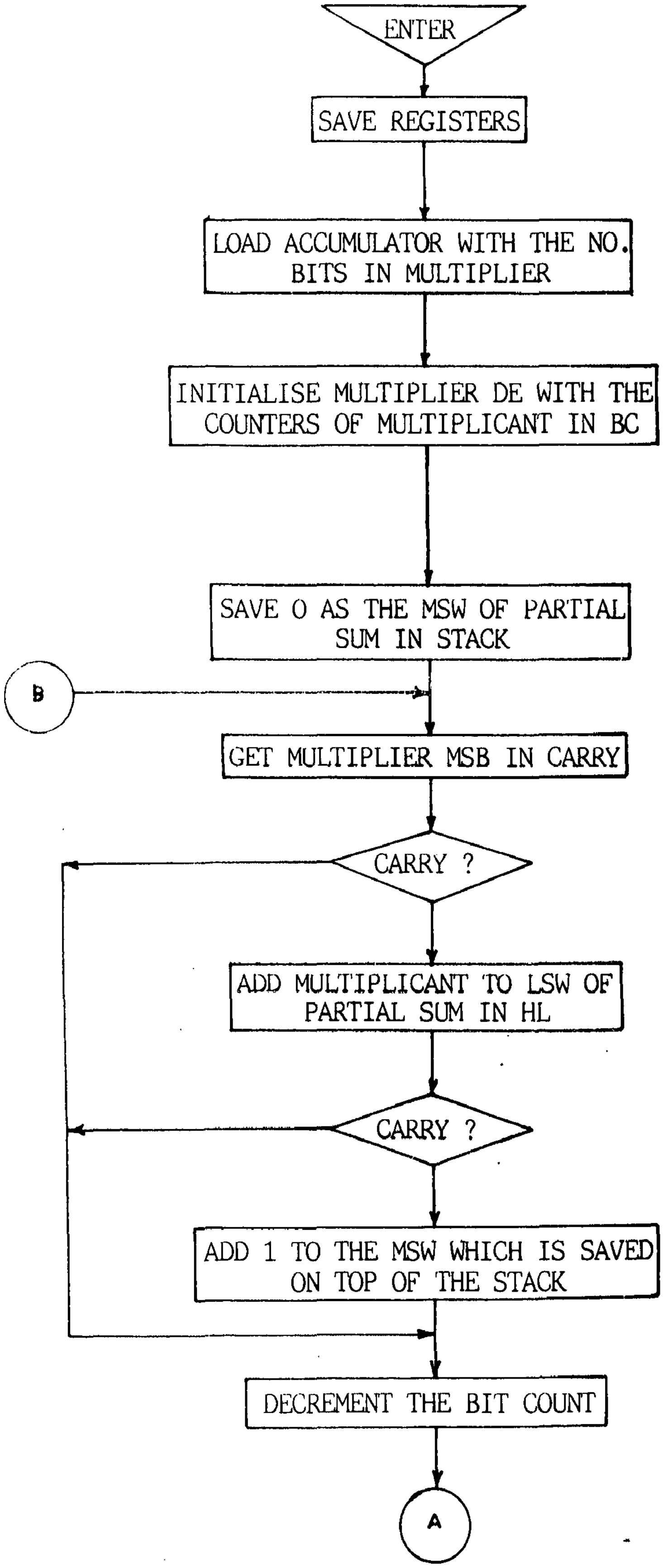


Fig.3.11 : Routine for square root calculation of a 32-Bit Number


ENTER
SAVE REGISTERS
LOAD ACCUMULATOR WITH THE NO. BITS IN MULTIPLIER
INITIALISE MULTIPLIER DE WITH THE COUNTERS OF MULTIPLICANT IN BC
SAVE 0 AS THE MSW OF PARTIAL SUM IN STACK
B
GET MULTIPLIER MSB IN CARRY
CARRY ?
ADD MULTIPLICANT TO LSW OF PARTIAL SUM IN HL
CARRY ?
ADD 1 TO THE MSW WHICH IS SAVED ON TOP OF THE STACK
DECREMENT THE BIT COUNT
A

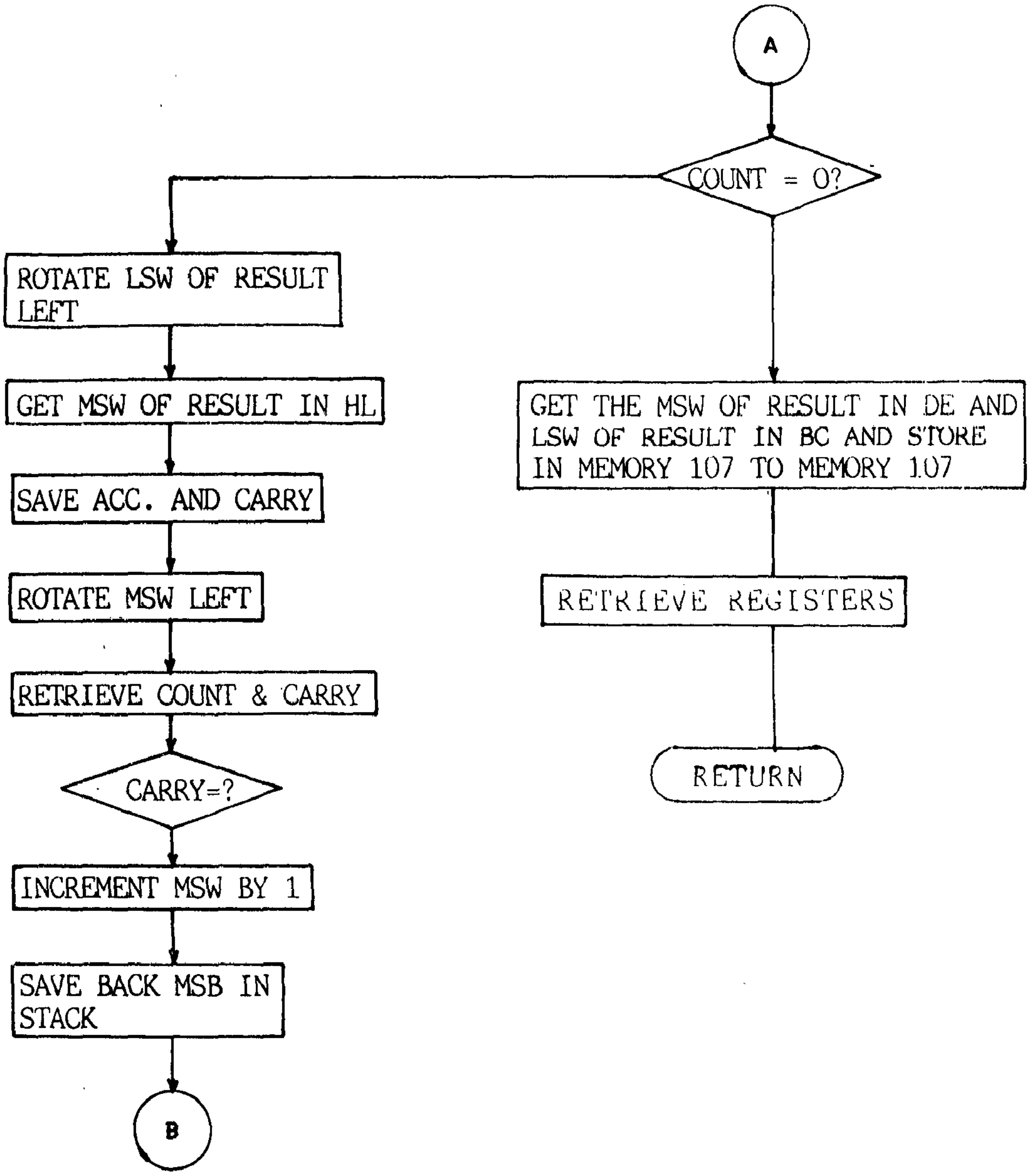


Fig.3.12 : Routine to square BC register pair

its square root is calculated by calling SQRT subroutine. Now the result of SQRT routine is divided by the square root of the total nos. i.e. it is divided bythe 04 decimal. The flow chart of this routine is shown in Fig.3.13.

n) Search Max No.:

This routine finds the Max Number (16-bit) from the N 16-bit numbers stored in sequential memory location. After getting the max no. from the block of data it compares it with its highest limit.If the max no. is exceeding higher limit it sets one corresponding bit in memory location BYTEMAX. The flow chart is shown in Fig.3.14.

o) Search Min No.:

This routine finds minnumber (16-bit) from the N 16-bit numbers stored in sequential memory locatioin. After getting the min no. from the block of data it compares it with its lowest values. If the min no. is lower than its lowest limit it sets one corresponding bit in memory location BYTEMIN. The logic used is the comparison logic. The flow chart is shown in Fig.3.15.

3.3.2 Application Example Software:

To show the use of data acquired and processed one example is coated. In this example 16 analog variables are subdivided into fast and slow variables. Fast variables are sampled at a rate of 1.25 ms and slow variables are sampled at a rate of 20 ms.

its square root is calculated by calling SQRT subroutine. Now the result of SQRT routine is divided by the square root of the total nos. i.e. it is divided bythe 04 decimal. The flow chart of this routine is shown in Fig.3.13.

n) Search Max No.:

This routine finds the Max Number (16-bit) from the N 16-bit numbers stored in sequential memory location. After getting the max no. from the block of data it compares it with its highest limit.If the max no. is exceeding higher limit it sets one corresponding bit in memory location BYTEMAX. The flow chart is shown in Fig.3.14.

o) Search Min No.:

This routine finds minnumber (16-bit) from the N 16-bit numbers stored in sequential memory locatioin. After getting the min no. from the block of data it compares it with its lowest values. If the min no. is lower than its lowest limit it sets one corresponding bit in memory location BYTEMIN. The logic used is the comparison logic. The flow chart is shown in Fig.3.15.

3.3.2 Application Example Software:

To show the use of data acquired and processed one example is coated. In this example 16 analog variables are subdivided into fast and slow variables. Fast variables are sampled at a rate of 1.25 ms and slow variables are sampled at a rate of 20 ms.

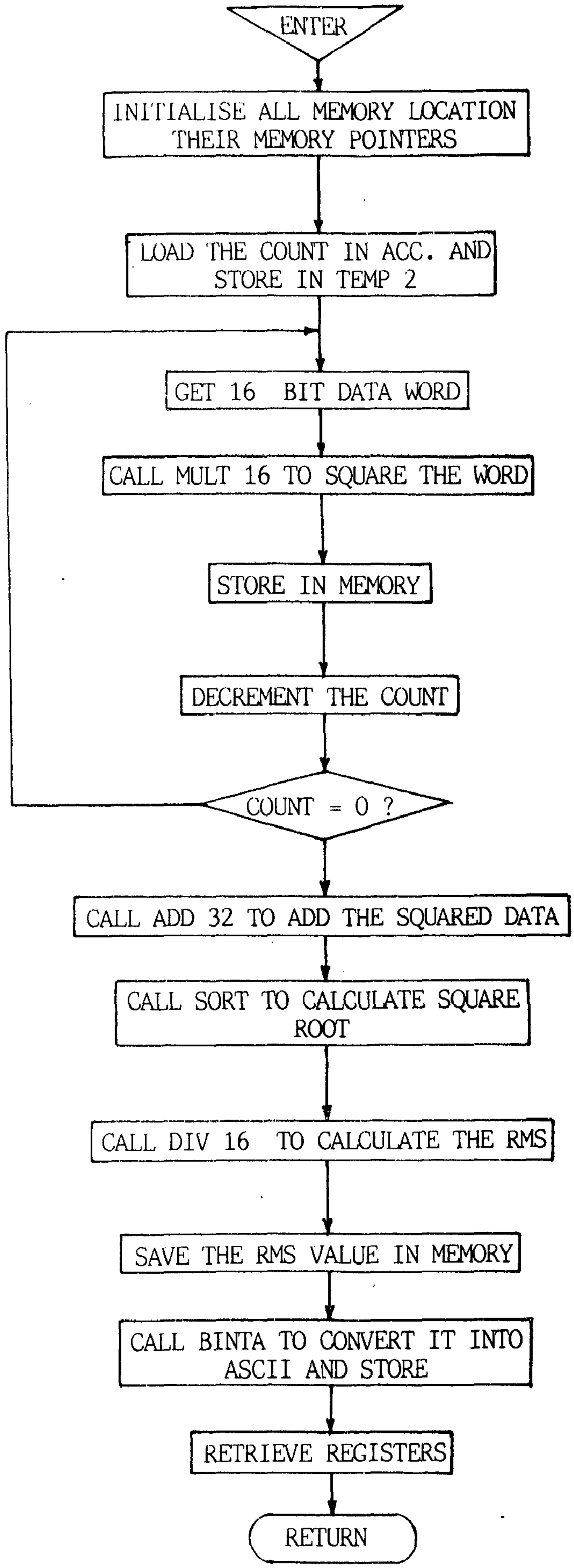
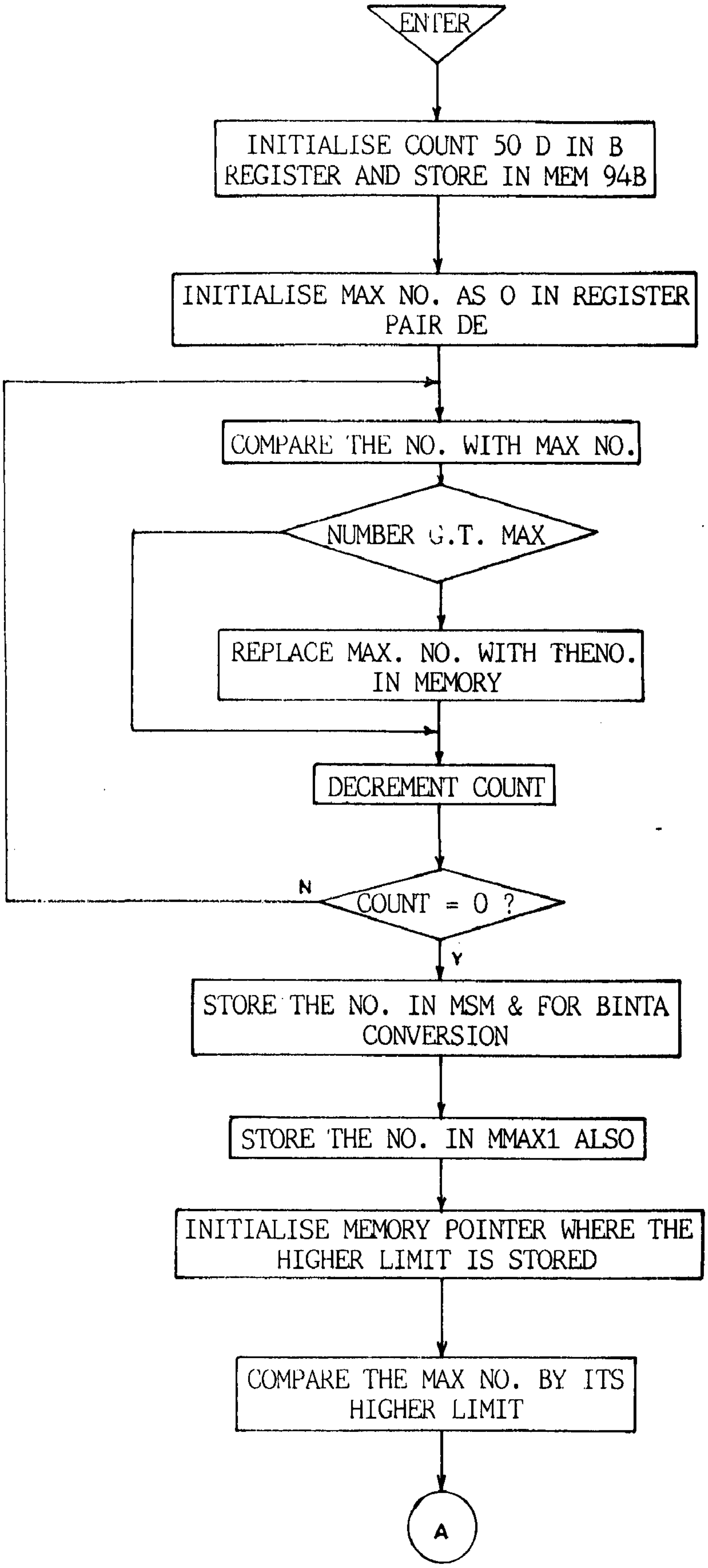


Fig.3.13 : Routine for RMS Calculation



ENTER
INITIALISE COUNT 50 D IN B REGISTER AND STORE IN MEM 94B
INITIALISE MAX NO. AS 0 IN REGISTER PAIR DE
COMPARE THE NO. WITH MAX NO.
NUMBER G.T. MAX
REPLACE MAX. NO. WITH THENO. IN MEMORY
DECREMENT COUNT
COUNT = 0 ?
N
Y
STORE THE NO. IN MSM & FOR BINTA CONVERSION
STORE THE NO. IN MMAX1 ALSO
INITIALISE MEMORY POINTER WHERE THE HIGHER LIMIT IS STORED
COMPARE THE MAX NO. BY ITS HIGHER LIMIT
A

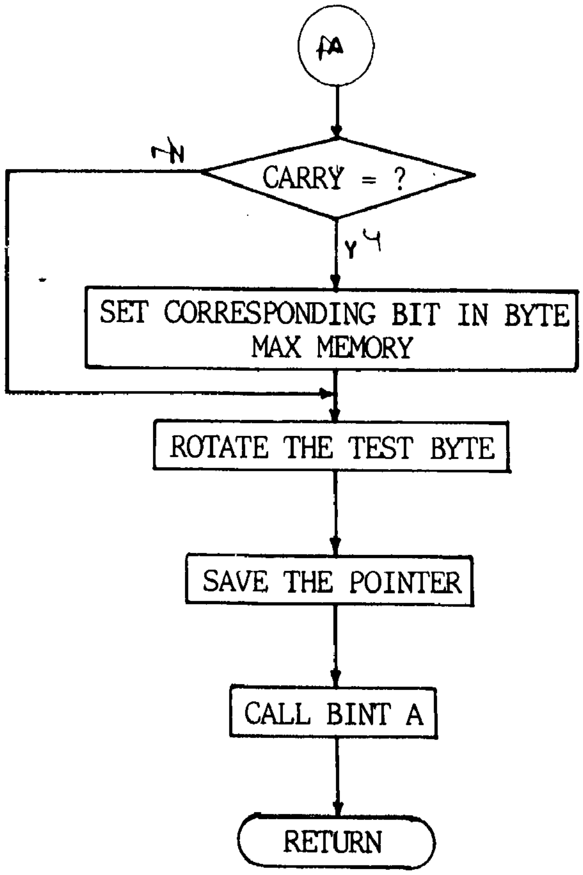
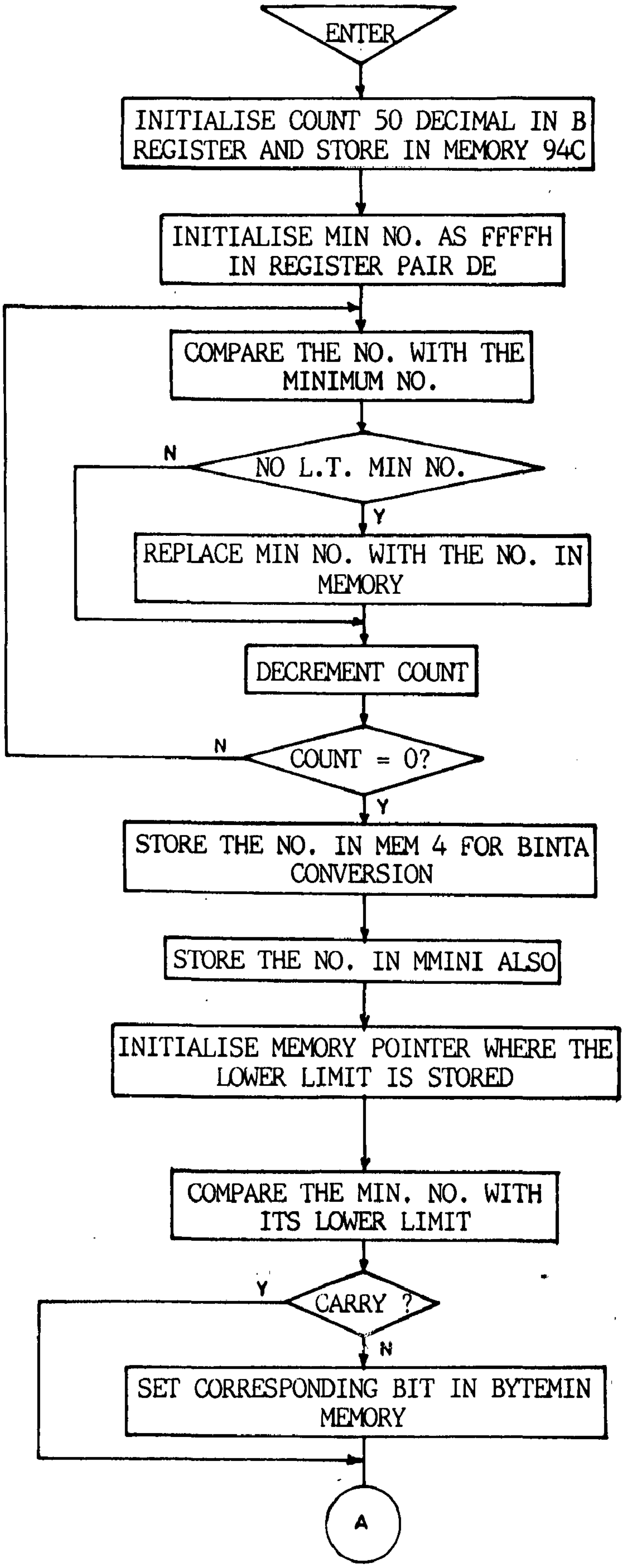


Fig.3.14 : Routine for finding maximum number in a string


ENTER
INITIALISE COUNT 50 DECIMAL IN B REGISTER AND STORE IN MEMORY 94C
INITIALISE MIN NO. AS FFFFH IN REGISTER PAIR DE
COMPARE THE NO. WITH THE MINIMUM NO.
NO L.T. MIN NO.
N
Y
REPLACE MIN NO. WITH THE NO. IN MEMORY
DECREMENT COUNT
COUNT = 0?
N
Y
STORE THE NO. IN MEM 4 FOR BINTA CONVERSION
STORE THE NO. IN MMINI ALSO
INITIALISE MEMORY POINTER WHERE THE LOWER LIMIT IS STORED
COMPARE THE MIN. NO. WITH ITS LOWER LIMIT
CARRY ?
Y
N
SET CORRESPONDING BIT IN BYTEMIN MEMORY
A

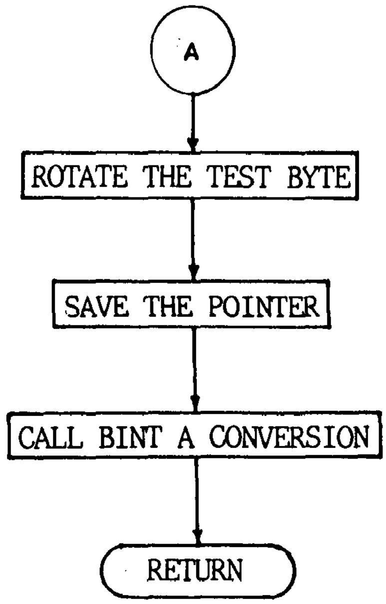


Fig. 3.15 : Routine to find Minimum Number in a String

There are four number of fast variable and 12 no. of slow variable, again in which 4 variables are controlled. These analog variables are the variable of a plant. There are 8-status inputs from switching devices like C.B etc. The status of switching devices are repeatedly sensed at a rate of 1 sec. To get an interval of 1.25 ms, interrupt RST 6-5 is used. The O/P of the counter 1 is connected to the RST 6.5. The counter 1 is initialised to mode 0 (interrupt on terminal count) and then loaded to give an interrupt after 1.25 ms. In ISS inspite of other work done in that routine the counter is again loaded to give again an interrupt after 1.25 ms, so in this manner repeated interrupts are generated after 1.25 ms.

Following routines and main program are used to simulate the example:

a) Main:

This main programs (shown in Fig.3.16) intialises all the interfacing chips, define public and external variables and initialise all the memory location. This will unmask all the three interrupts (RST 7.5,RST 6.5, RST 5.5) and then inputs all the 16-analog variables and store their value in memory. Analog variables inputted by call slow and fast subroutine. Load the counter 1 with 1.25 ms count and waits for an interrupt.

b) Slow:

This subroutine (shown in Fig.3.17) inputs the value of four analog controlled variable (01 to 04) and store their value in

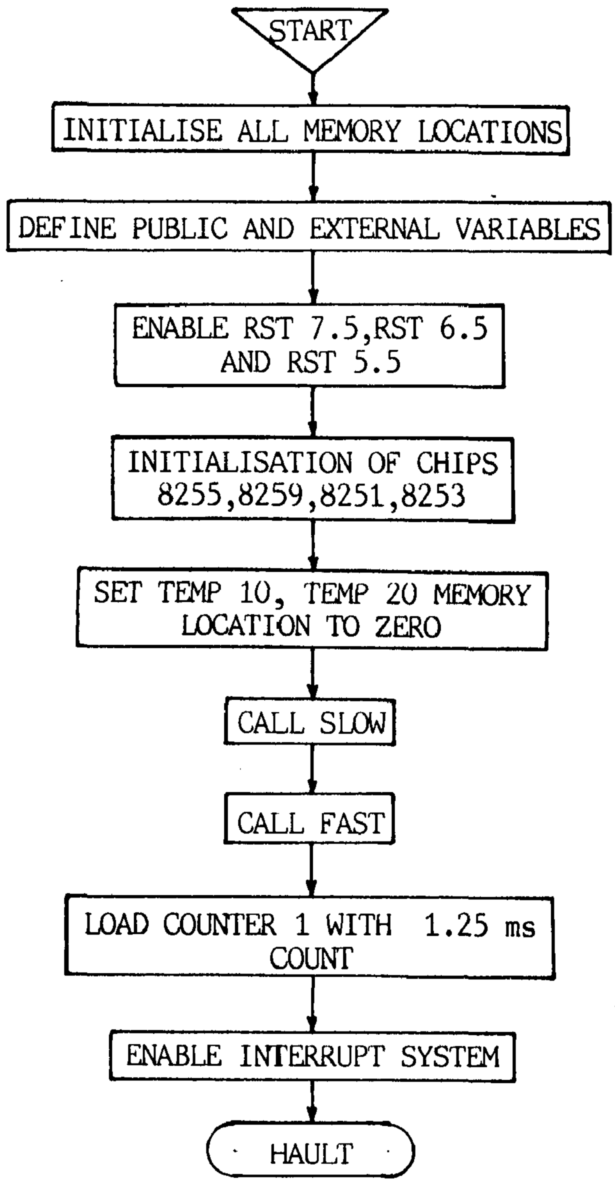


Fig.3.16 : Routine to input fast and slow analog variables

A

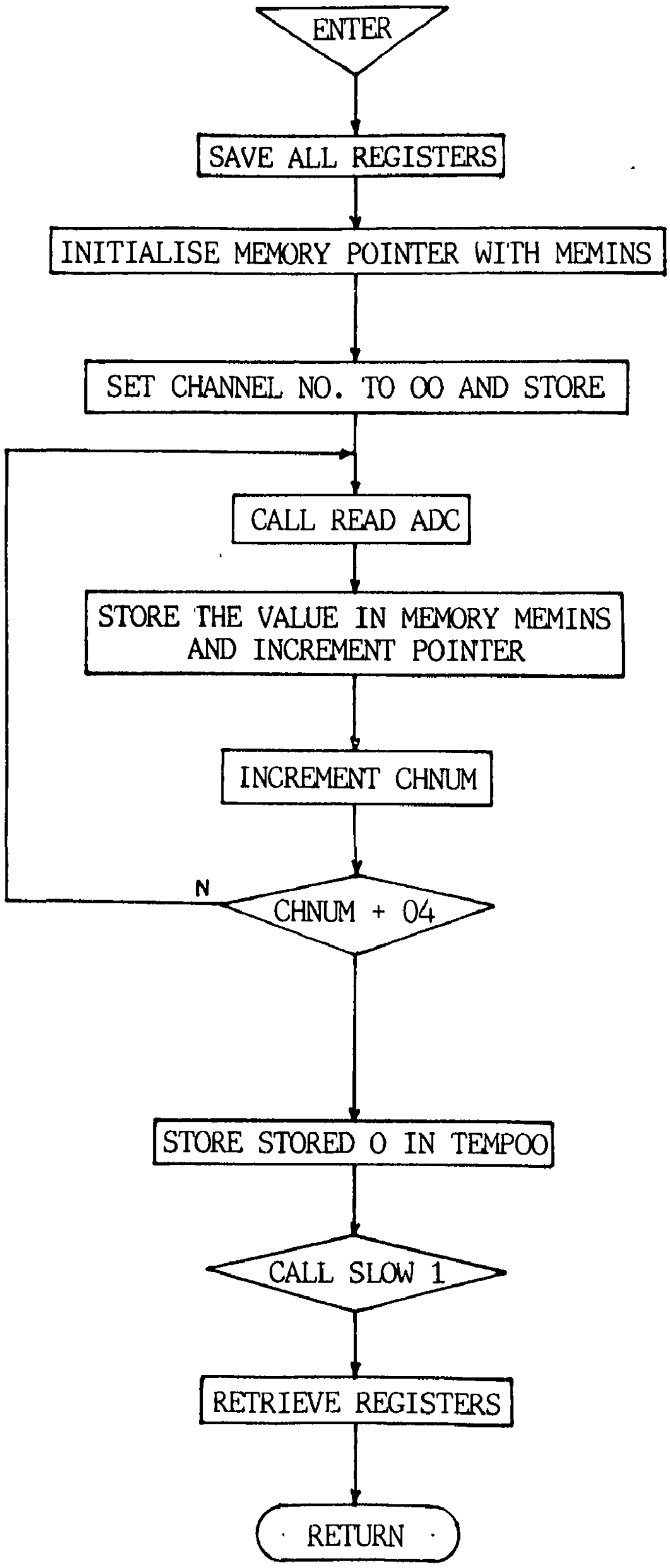


Fig.3.17 : Routine to input 4 analog variables (01,02,03,04) (SLOW)

memory. These values are also converted into ASCII to directly display on CRT. Within this routine SLOW1 is called.

c) Slow 1:

This subroutine (shown in Fig.3.18) inputs the value of 8 analog uncontrolled variable (05 to 12) and store them in memory.

d) Fast 1:

This subroutine (shown in Fig.3.19) inputs the value of 4 analog variable (13-16) and store them in memory.

e) Interrupt service subroutine for RST 6.5:

The following functions are performed in this routine. The flow chart is shown in Fig.3.20.

1) It reloads the counter 1 with 1.25 ms count.

2) It checks if the one cycle of 50 Hz is over by checking if the 16 samples of fast variables are taken.

2a) If 16 samples are over, it calculates average and rms values of those variable and store in memory after converting them into ASCII. And increment the count for slow variables and input slow variables. If 20 ms period i.e. 16 samples are not over it inputs the value of fast variable by calling fast subroutine and returns to the main program.

3) It checks if the 50 samples of slow variables are over. If the 50 samples are not over, it repeats the step 2(a). If the 50 samples i.e. 1 sec period is over, it calculates the average, Max, Min values for variable (05-12) and again calculate rms

A

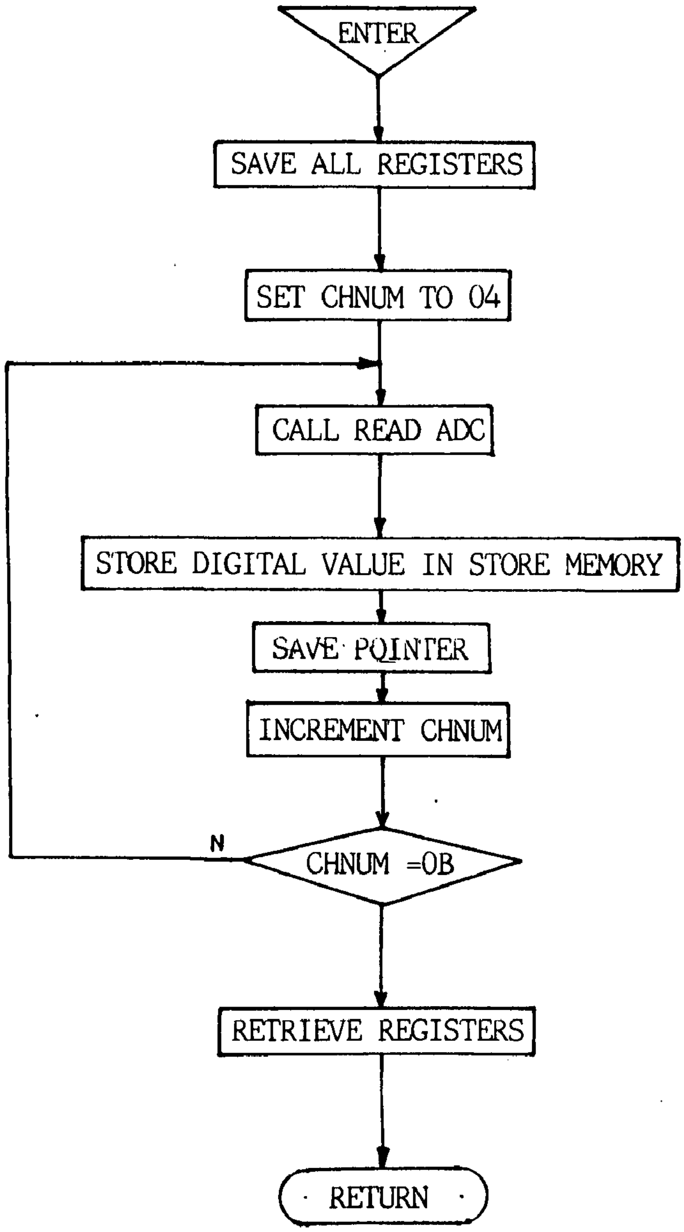


Fig.3.18 : Routine to input 8 analog Variables (05-12) (SLOW)

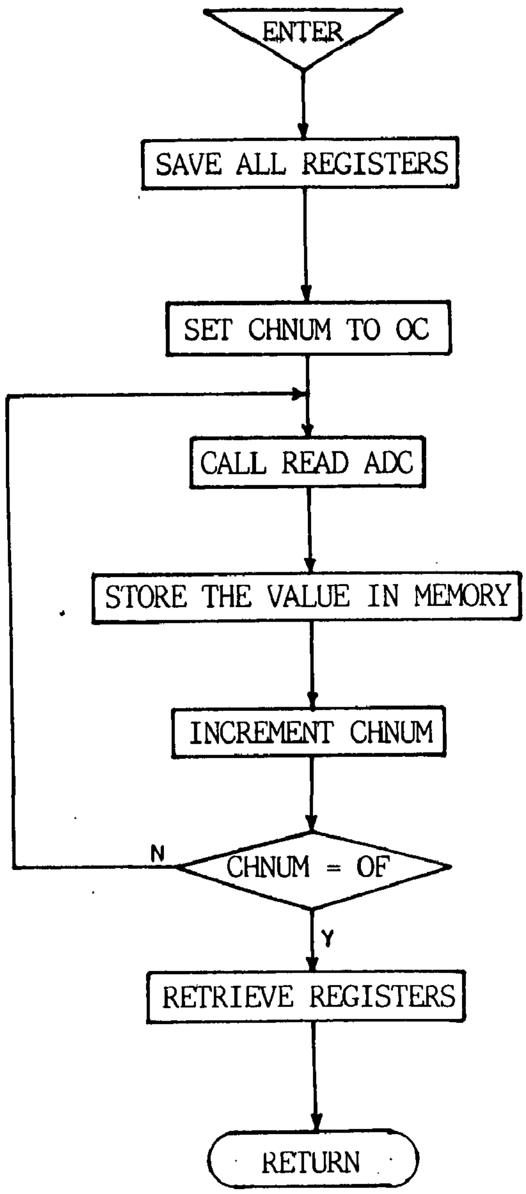
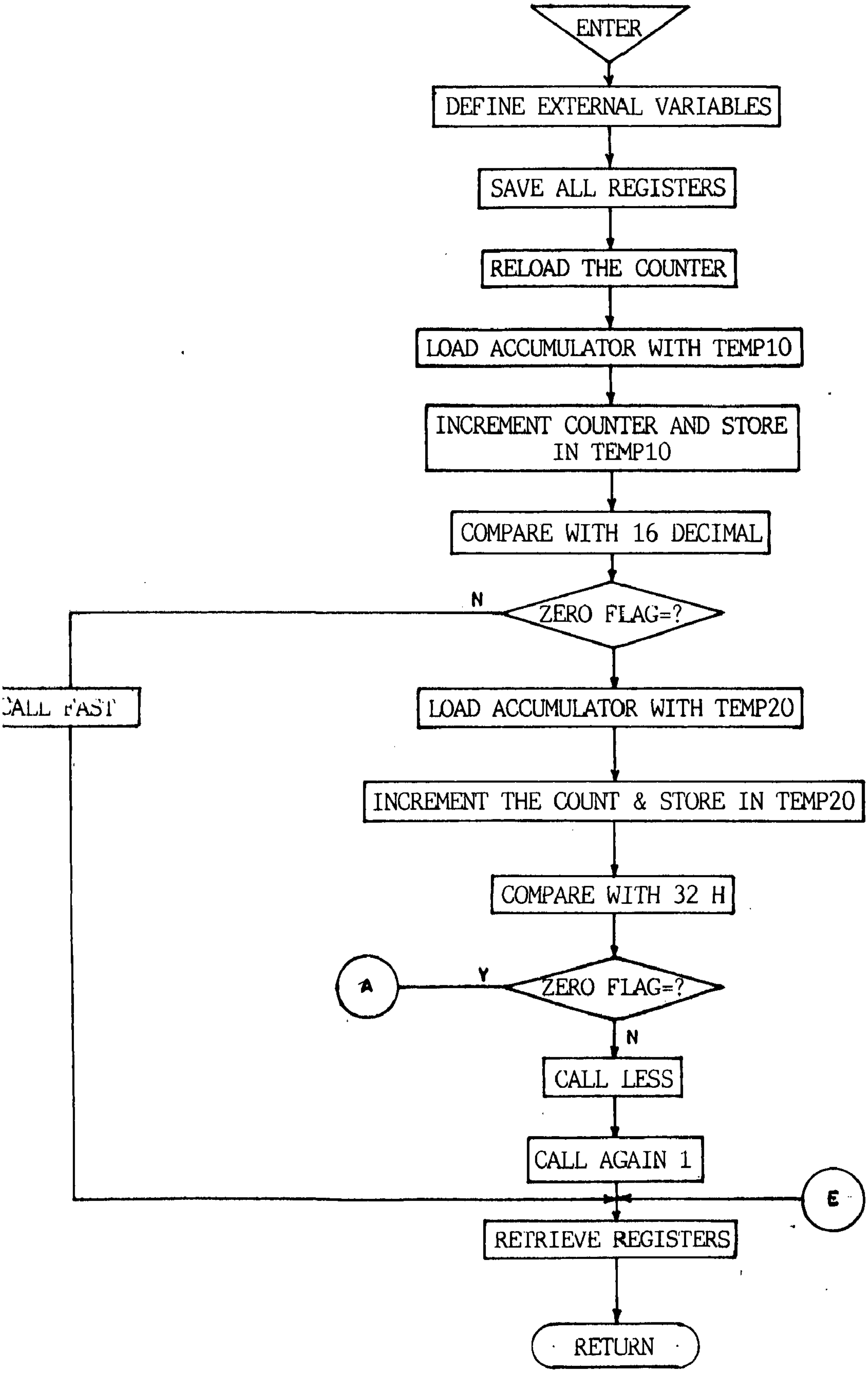


Fig. 3.19 : Routine to input 4 analog variable(OC-OF) (FAST)


ENTER
DEFINE EXTERNAL VARIABLES
SAVE ALL REGISTERS
RELOAD THE COUNTER
LOAD ACCUMULATOR WITH TEMP10
INCREMENT COUNTER AND STORE IN TEMP10
COMPARE WITH 16 DECIMAL
ZERO FLAG=?
N
CALL FAST
LOAD ACCUMULATOR WITH TEMP20
INCREMENT THE COUNT & STORE IN TEMP20
COMPARE WITH 32 H
A
Y
ZERO FLAG=?
N
CALL LESS
CALL AGAIN 1
E
RETRIEVE REGISTERS
RETURN

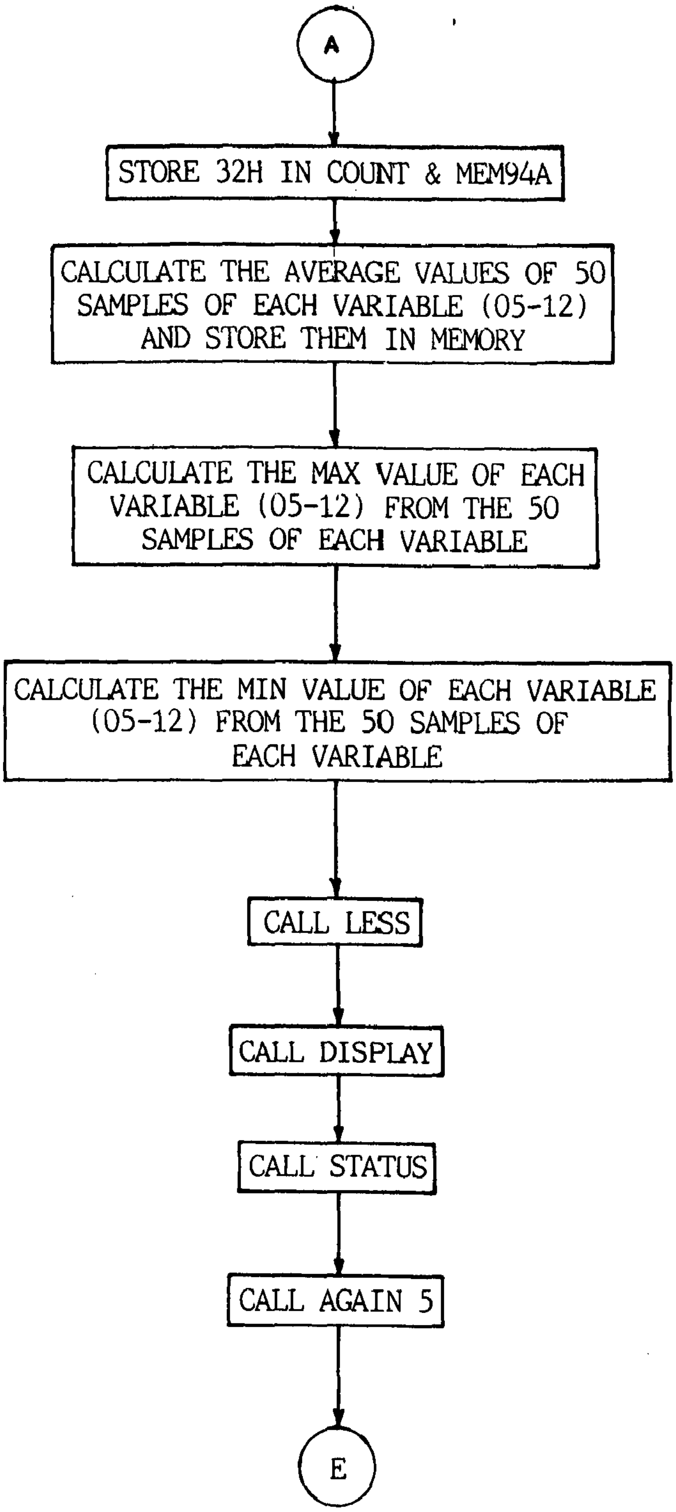


Fig.3.20 : Routine to Input, Process and Display Analog and digital variables (INT 65)

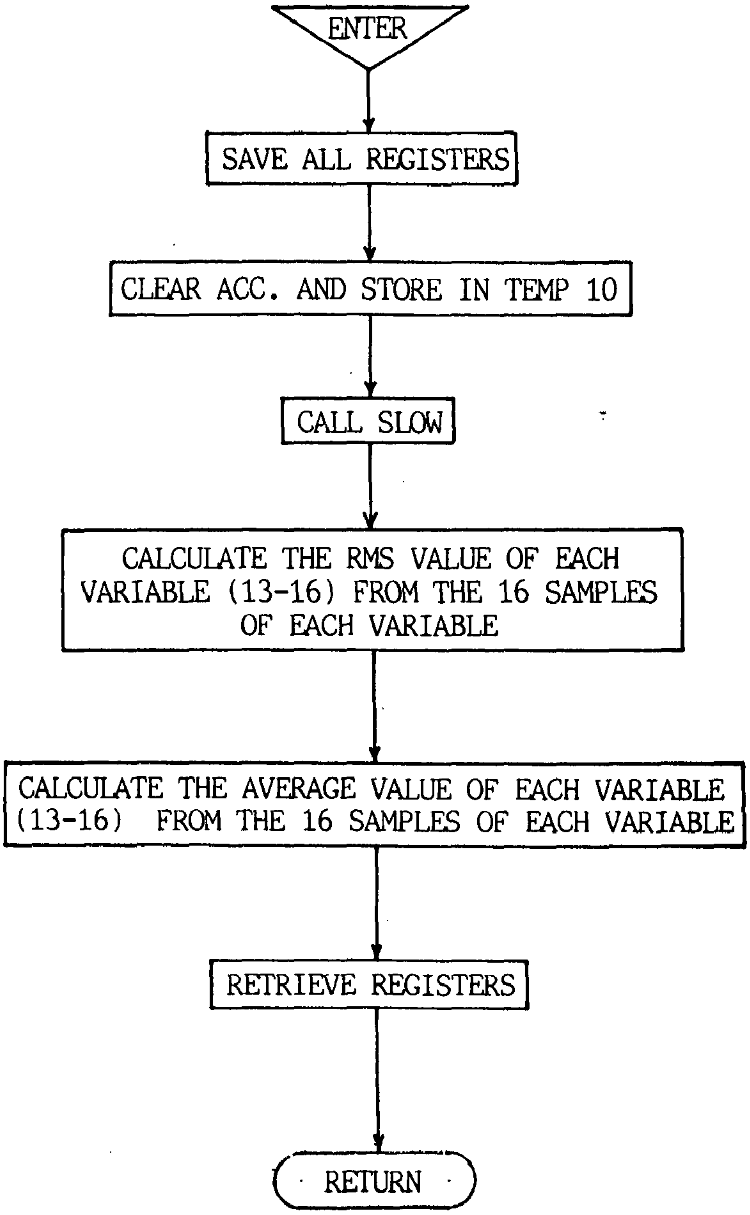


Fig.3.21 Routine to calculate RMS and Average Values for Fast Variables

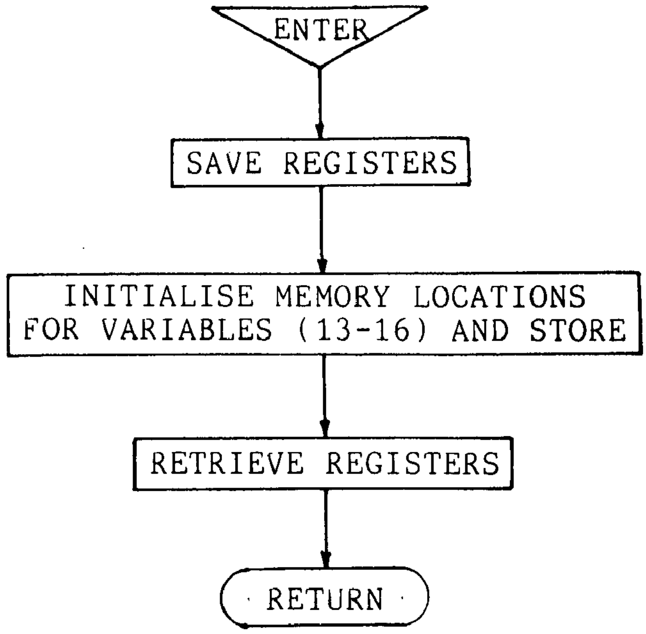


Fig.3.22 : Routine for Initialisation of Memory Location for Fast Variables (Again 4)

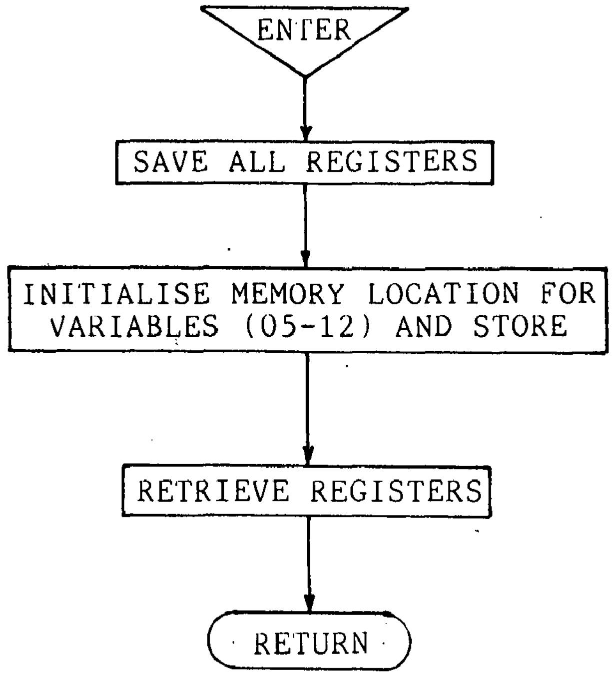


Fig.3.23 : Routine for Initialisation of Memory Location for slow variables (Again 5)

value for fast variables (13-16). After the whole processing it calls display routine to display the whole processed data on CRT. It also calls status routine will inputs the status of 8-switching device and displays on CRT their ON & OFF position. Then it returns to the main program.

In this manner the application example is developed to show the use of various inputs.

3.3.3 Display Software:

The display software manages to display the information shown in Table 3.1 on CRT. In the table it self the memory location where the codes (IN ASCII) are stored for each line is also shown. The whole block (B000 H to B45FH) is used to store this information. To achieve this first the whole block is filled with the ASCII code of blank (20) then from the program itself the value for instantaneous, maximum, minimum, average, RMS comes which are stored in proper memory location. The status of 8 switching devices are also sensed and it is also stored in proper memory location. The constant data for example codes for variable No. etc. are permanently stored in memory location. When the whole block is filled with ASCII data bytes then following two routines are called to transmit the information on CRT.

a) Display Analog Information :

The routine flow chart is shown in Fig.3.24. This routine initialises the counter with the length of memory block and initia-

Corresponding Memory Location					
B000	VA				
B050	RE				
B0A0	IN				
B0F0					
B140	VA				
B190	MA	10	11	12	
B1E0	MI	XXX	XXXX	XXXX	
B230	AV	"	"	"	
B280		"	"	"	
B2D0	VA				
B320	RM				
B376					
B3C0	SW				
B410	ST	06	07	08	
		X	X	X	(ON OR OFF)

DISPLAY ON CRT

Corresponding Memory Location											
B000	VARIABLES	NO.	01	02	03	04					
B050	REFERENCE		XXXX	XXXX	XXXX	XXXX					
B0A0	INSTAWTNEOU		XXXX	XXXX	XXXX	XXXX					
B0F0											
B140	VARIABLE	NO.	05	06	07	08	09	10	11	12	
B190	MAXIMUM	XXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	
B1E0	MINIMUM		"	"	"	"	"	"	"	"	
B230	AVARAGE		"	"	"	"	"	"	"	"	
B280											
B2D0	VARIABLE	NO.	13	14	15	16					
B320	RMS		XXXX	XXXX	XXXX	XXXX					
B376											
B3C0	SWT DEVICE	NO.	01	02	03	04	05	06	07	08	
B410	STATUS		X	X	X	X	X	X	X	X	(ON OR OFF)

TABLE 3.1 INFORMATION DISPLAY ON CRT

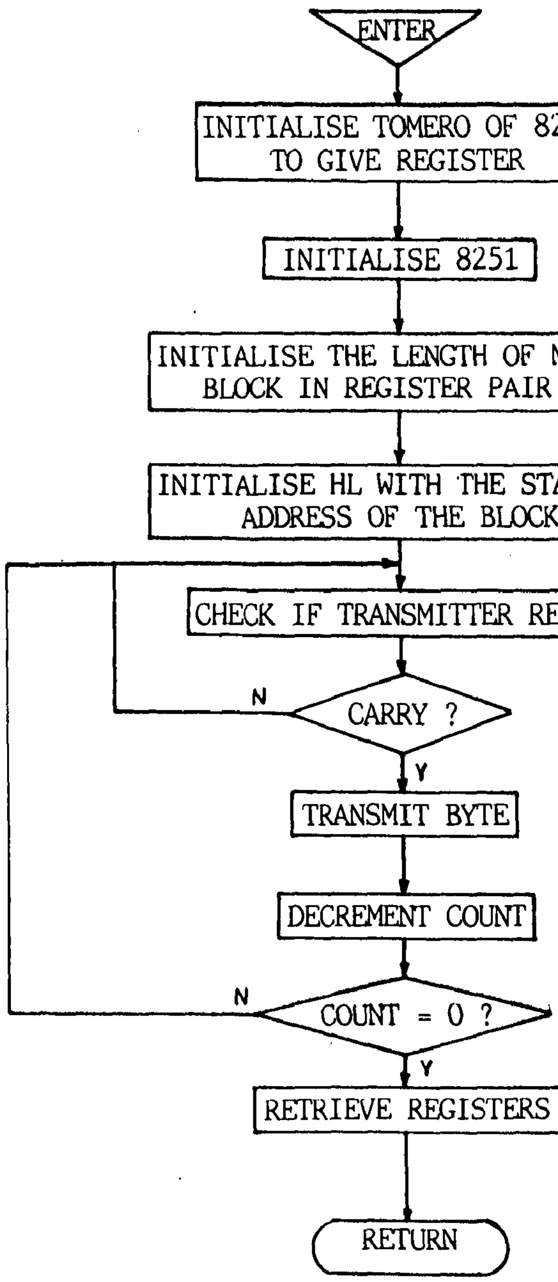


Fig.3.24 : Routine to display a block of memory on CRT

lises the memory pointer with the starting address of memory block. After this it checks if the transmitter of 8251 is ready to transmit byte, when it is ready it transmits one byte and in this manner the whole information is transmitted. The starting address of memory block is B000H.

b) Display Digital Information :

The flow chart for this subroutine is shown in Fig.3.25. This routine first inputs the status of 8-switching deviced from 8255 and checks the status of devices (ON OR OFF) and stores the corresponding code in memory. The status is checked bybit shifting. After the status of all 8-devices are checked the information is displayed on CRT. The starting address of memory block is B3C0H.

3.3.4 Communication Software:

The communication with MCS is achieved by using a definite protocol and according to that protocol software for RTU is developed which is discussed in Chapter 5 on communication.

3.3.5 Control Software:

a) PID Controller :

The control signal O/P of PID controller is as follows :

$$mlt = K_p e - K_D \frac{d_e}{d_t} + K_I \sum_{o}^{t} e\, dt + M_o$$

e = r-b

r = reference value

p = true or instantaneous value

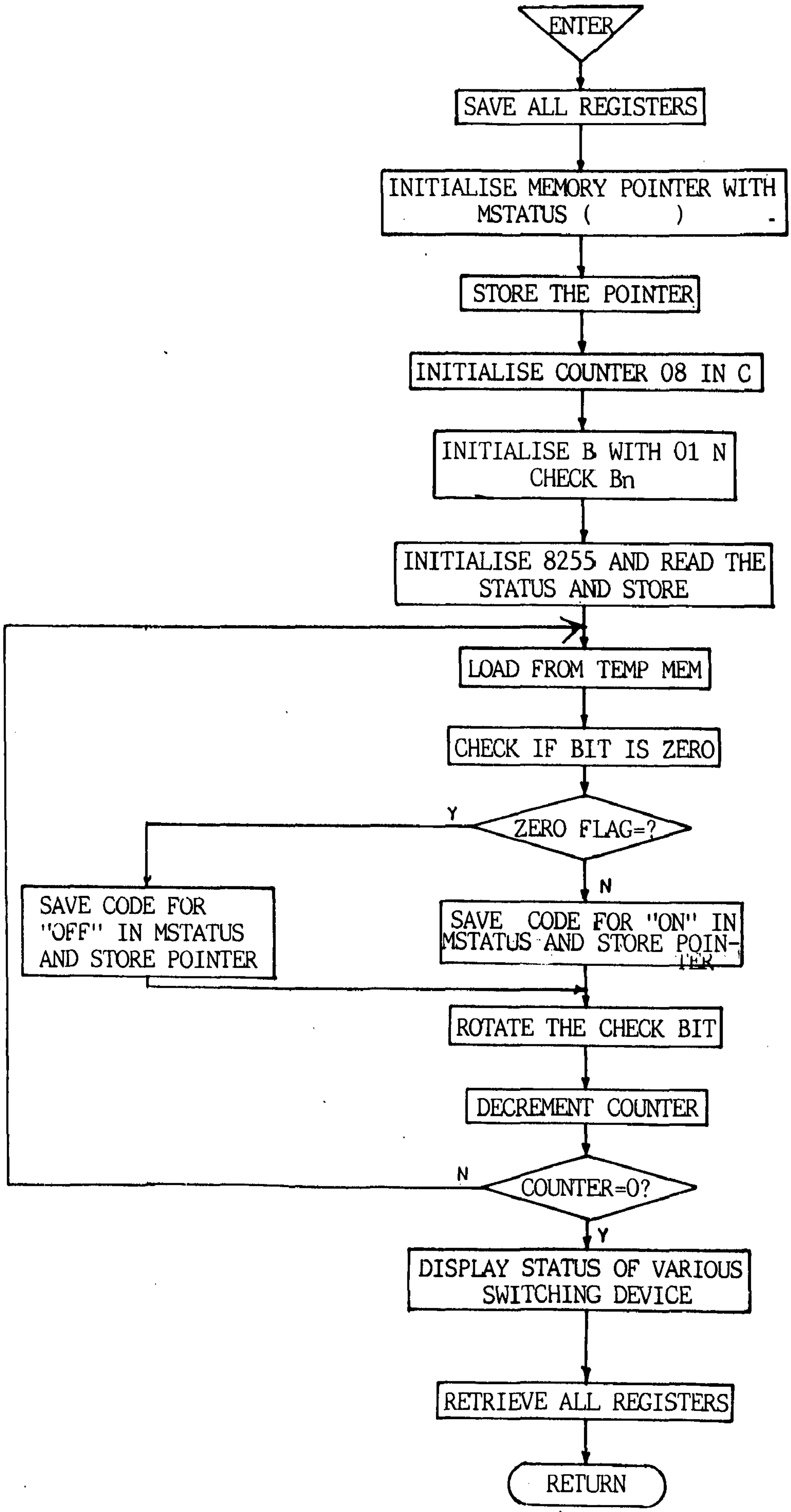


Fig..3.25 Routine to display status of switching

e = error signal

M_o = initial value when no. error

t = sampling interval

K_p = proportional constant

$K_D = T_D$ = differentiation constant

$K_1 = \frac{1}{T_1} =$ Integration constant

$$m_n = K_p e_n + K_I \sum_{j=0}^{n} e_j t + K_D \frac{e}{Dt} + M_o$$

$$m_n = K_p e_n + K_I \sum_{j=0}^{n} e_f + K_D (e_n - e_{n-1}) + M_o \qquad (1)$$

four constants K_p, K_I', K_D' and M_o are stored in memory. The equation (1) is simulated for PID control action and its flow chart is shown in Fig.3.26.

b) Control Commands via Interrupts :

Two control commands as follows :

1) Stop switching device

2) Stop plant

Command comes via two interrupt of 8259 (programmable interrupt controller). These interrupts are IRO, IR1. Each command can come from eight control/protection units. The eight signals for each command are ORed and the output of OR gate is given to the interrupt pins of 8259. These control commands are sensed and then displayed

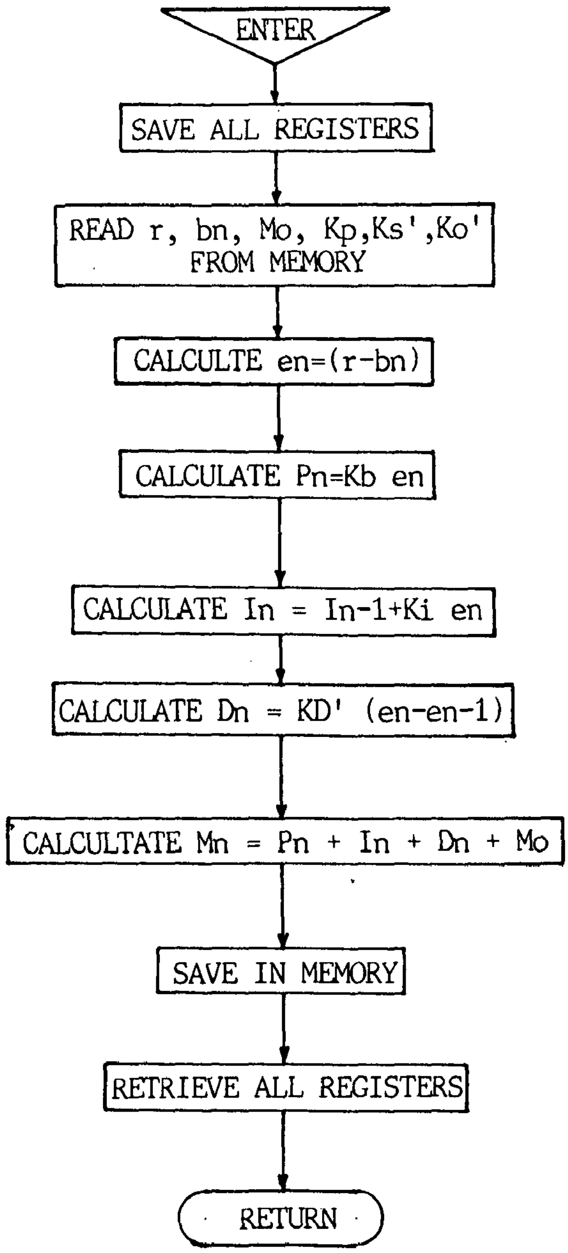


Fig.3.26.1 Routine to calculate Mn (MORE)

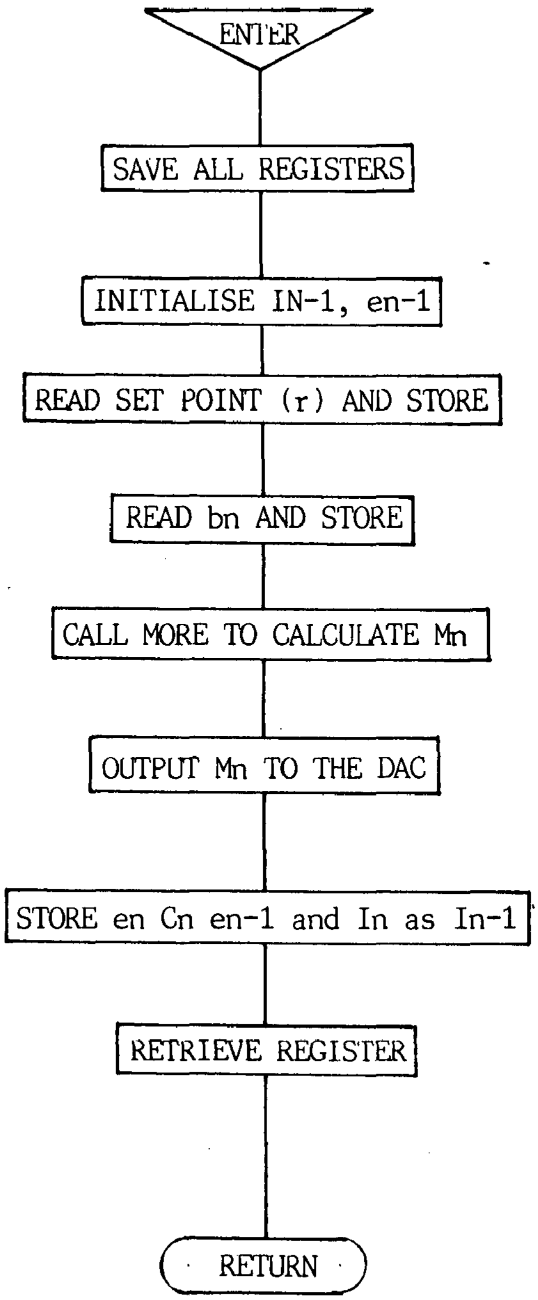


Fig.3.26 : Routine for PID Controller

on CRT. The two routines developed for individual commands are shown in Fig.3.27. The logic used to display the commands are same but only the contents of memory location changes according to the control command.

Display Control Command:

The two routine named IRO, IR1 are developed according to the control command interrupt. The logic to develop these routines is shown in Fig.3.27.

In these two interrupt service subroutine, the status of 8 inputs for each routine are sensed through ports of 8255 (programmable peripheral interface) and then by checking the status of each bit it is known, from which devices or plant control commands are coming. After sensing the status of various devices display subroutine (DISP) is called to display the control command.

Display Information (DISP):

This routine (shown in Fig.3.28) transmits the byte to CRT stored from a particular memory location. For interrupt IRO, ASCII data is stored from B500, ASCII data for IR2 is stored from B600 memory location.

B500	STOP SWT. DEVICE NO.
B600	STOP PLANT NO.

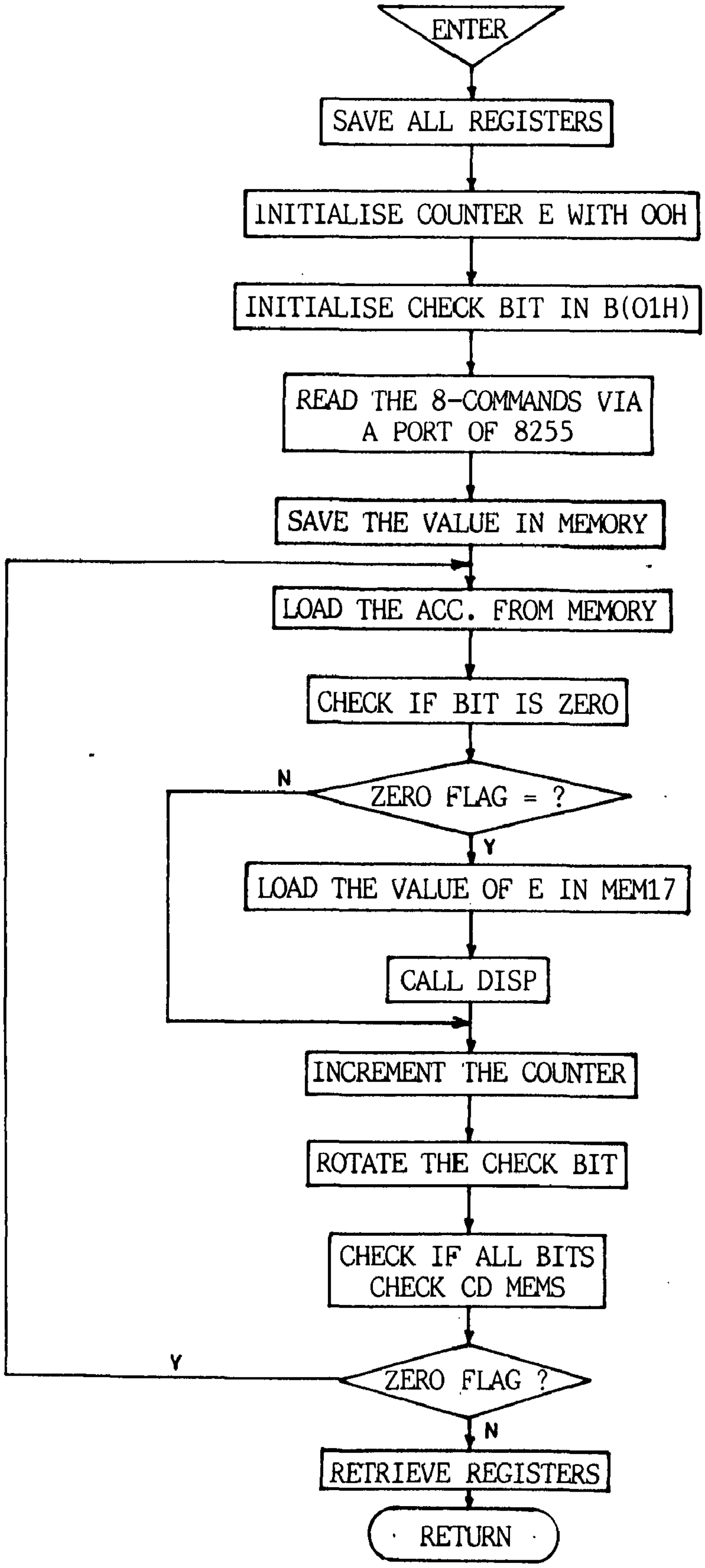


Fig.3.27 : Routine which checks various commands and displays

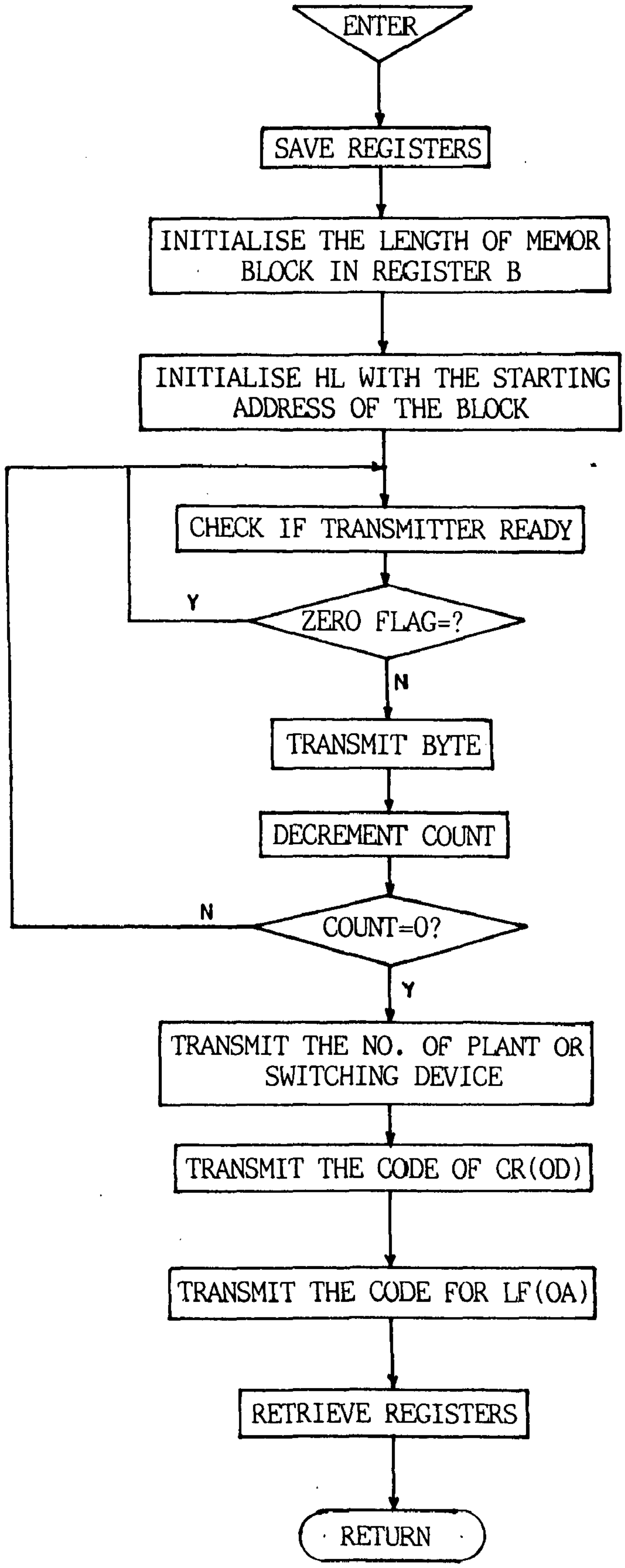


Fig.3.28 : Routine to display the control command

Before displaying control commands on CRT, the screen is cleared via BLANK OUT CODES then these commands are displayed. After the display of command display of analog and digital information is also done by calling their routines. The codes for blanking OUT CRT screen are [1B,5B,4B,5B,32,4A]. The blocking out of screen is done to display commands more clearly.

c) Control Commands via Keyboard:

The control command given by MCS to RTU can also be given in RTU by operator. The operator can gives following commands via key-board.

CODE	FUNCTION OF THE CODE
1	START PLANT
2	STOP PLANT
3	START DEVICE
4	STOP DEVICE
5	REVIEW REFERENCE VALUE
6	CHANGE HIGHER LIMIT
7	CHANGE LOWER LIMIT
8	RESET COUNTER 0
9	RESET COUNTER 1
A	READ COUNTER 0
B	READ COUNTER 1

If any other code is pressed, than it will display "WRONG CODE PRESSED". The logic for these control commands are the same as used for MCS control commands, the only difference is for MCS control commands values are provided by MCS and in these the values for different commands are given via keyboard. The details of these codes are discussed in Chapter-5.

These commands are useful in case of any communication failure occurs and if the operator wants to override the MCS then by using these commands operator can perform its job.

CHAPTER-4
MASTER CONTROL STATION

This chapter discusses the details of master control station (MCS) of our system. WIPRO 80286 based PC is used for hardware of MCS. With the help of different hardware modules of PC various facilities of SCADA on MCS side are achieved. The hardware and software to achieve these facilities are described.

4.1 FACILITIES PROVIDED;

The following facilities are provided in the present MCS.

1. Collection of information from RTU's and displaying it in the form of MIMIC diagram.
2. Control command from MCS: Different control commands are given to RTU's via operator's key board on MCS. These commands are read by the CPU of MCS and sends that to RTU.
3. Hard copy : A printer is used for getting hard copy of the information with in the MCS.

4.2 HARD WARE:

Various hardware modules are used for achieving different facilities of MCS, shown in the form of a block diagram in Fig. 4.1. Two communication ports 1 and 2 are provided through 8251 (universal synchronous asynchronous receiver transmitter). Two different RTU's are connected by these communication ports. The communication between MCS and RTU is a serial communication. For

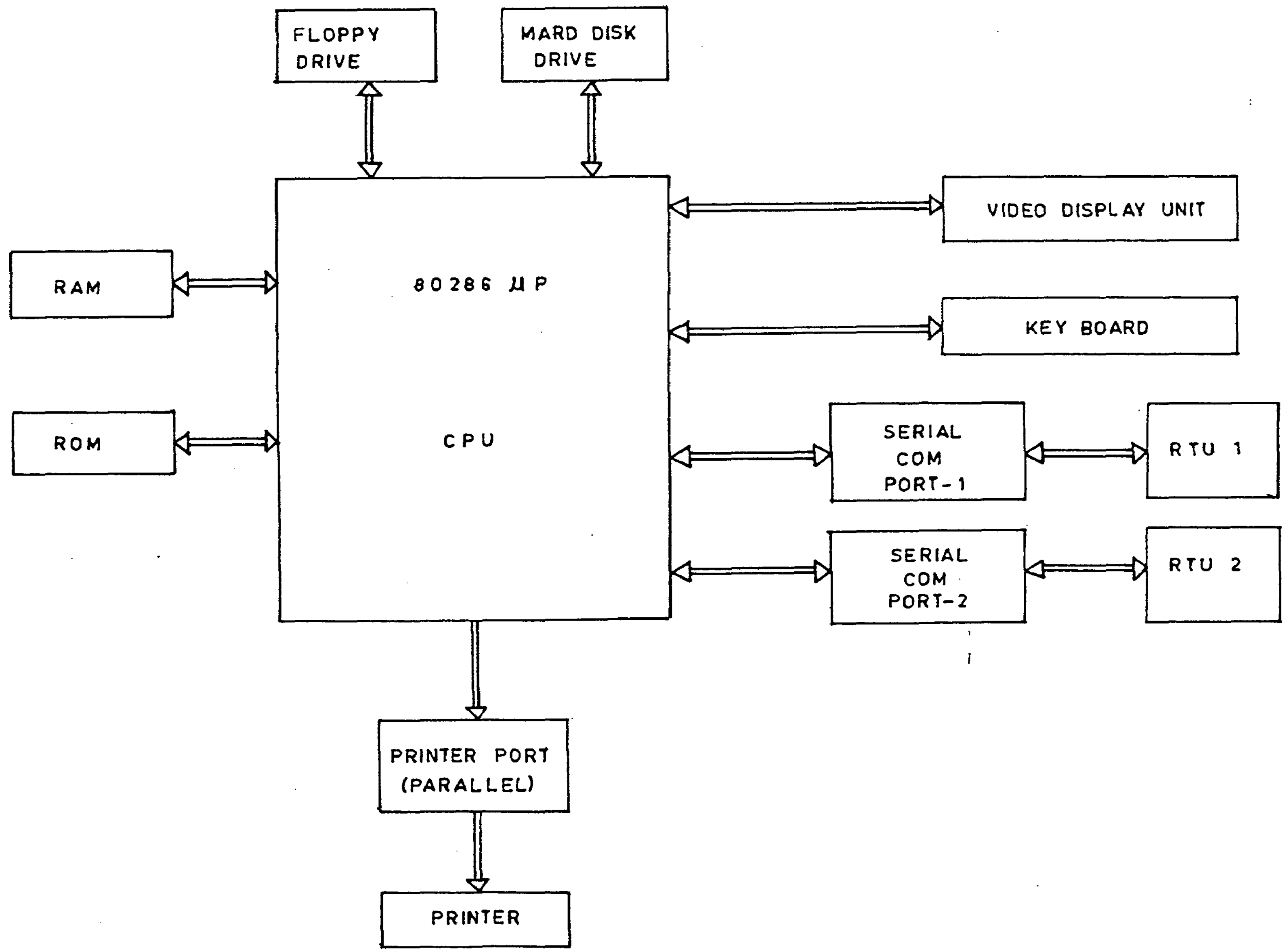


FIG. 4·1 BLOCK DIAGRAM OF MCS

the display of information one video display unit is linked with 80286 up. The information displayed can be graphical as well as in text or tabular form. For the permanent storage hard disc is used. This disc contains the operating systemand users software. For transporting software, a floppy disc drive is incorporated. RAM is used to store the programs transferes from hard disc and to execute them. ROM contains some of the monitor program. One parallel printer is provided for obtaining hard copy of any display or software when required.

4.3 SOFTWARE OF MCS:

MCS software is divided into 3 parts on the basis of the use as below :

4.3.1 Communication of Software:

Protocol for the communication is described in Chapter-5. The software for the MCS end has been written in the assembly language of 8086.

4.3.2 Control Software:

Control commands to be sent to RTU can be given by operator through keyboard of MCS. The PC reads the key board by using INT 16H of DOS. The function of INT 16 is described as follows[11].

i) <u>Interrupt:</u>

16H keyboard I/O

ii) Function Request:

OOH Read next keyboard character

To activate this function request AH should contain OOH and AL will contain the keyboard character pressed.

iii) Description:

This function request reads a character type at the keyboard. If the character has already been typed, and resides in the key board puffer, the character is returned immediately. Otherwise, this function request waits until a character is typed.

This function request returns with the ASCII code of the character typed in AL.

In this way the command inputted byoperator is registered in PC and which is further used.

Various Control Command:

Type	Function
0	SEND PREASSEMBLED PACKET
1	START PLANT
2	STOP PLANT
3	START DEVICE
4	STOP DEVICE
5	RENEW REFERENCE VALUE
6	CHANGE HIGHER LIMIT
7	CHANGE LOWER LIMIT
8	RESET COUNTER 0

9 RESET COUNTER 1

A READ COUNTER 0

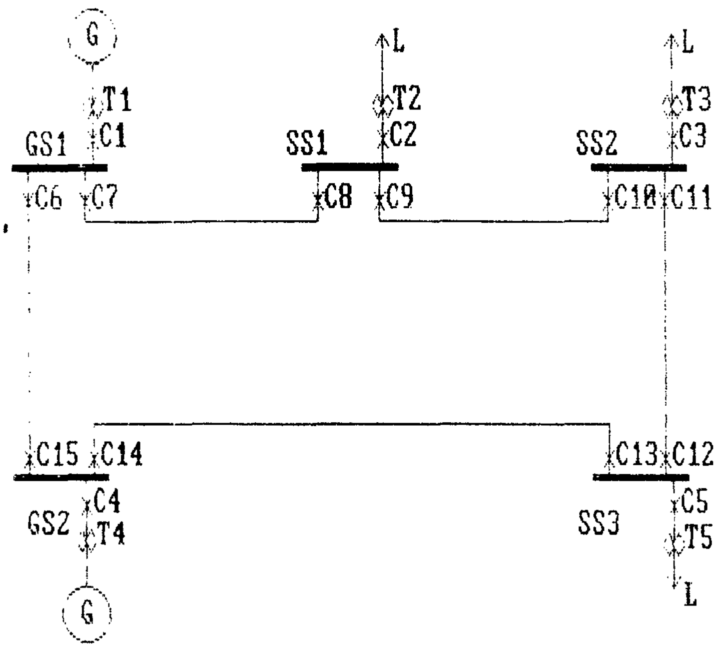
B READ COUNTER 1

If any other than these codes is pressed an alarm is given in the form of a keep.

The plant and device number to be stopped or started is input through keyboard. Similarly the reference, higher and lower limits are also input through keyboard. This control software has also been written in assembly langauge of 8086.

4.3.3 Display software:

The display includes a mimic diagram of the system being controlled and numerical information in tabular form. The mimic is made using the graphic software package GRAPH-X (Details given in the Appendix-F). The display software has been written in Fortran-77. An example of display is given in Fig.4.2.



CIRCUIT BREAKER STATUS

CB1	ON	CB9	OFF
CB2	ON	CB10	OFF
CB3	ON	CB11	ON
CB4	ON	CB12	ON
CB5	ON	CB13	ON
CB6	ON	CB14	ON
CB7	ON	CB15	ON
CB8	ON		

BUS VOLTAGE(KV)

GS1	132
GS2	132
SS1	132
SS2	132
SS3	132

	VOLTAGE(KV)		POWER(MW)		MAX TEMP(C)	
	REF	INSTA	REF	INSTA	BEARING	WINDING
GEN1	11	10.8	100	101	80	60
GEN2	15	10.2	200	198	85	65

DIS. VOLTAGE(KV)

SS1	33
SS2	11
SS3	11

CHAPTER - 5
COMMUNICATION

In this chapter the link used for communication between Remote Terminal Unit(RTU) and Master Control Station(MCS) is described. This is followed by a detailed discussion on the protocol used for the exchange of information between the two stations. Data structure for communication between the two stations is discussed and finally the software for Master Control Station and Remote Terminal Unit are presented.

5.1 COMMUNICATION LINK:

The communication link used between MCS and RTU is an RS232C link. This is a serial communication link. It is achieved by connecting only three lines (TXD, RXD & GND) between the 8251(Universal Synchronous Asynchronous Receiver Transmitter) IC's of the two stations with 3 wires. Following format of serial communication is used.

No. of start bits	=	1
No. of data bits	=	8
Parity bit	=	1
Stop bit	=	1

For the transmission of one byte of data, the transmitter of 8251 formats the data byte by adding start, parity and stop bits. The number of bits in the formatted data is eleven and transmission is done at a rate of 4800 bauds.

5.2. COMMUNICATION PROTOCOL:

The protocol for communication between MCS & RTU is similar for both MCS & RTU. The system that wants to send a message has to go through the following steps. In the present system requests always go from MCS to RTU at an interval of every one second.

1) MCS sends a request in the form of "Identity Number" of RTU and then enters a time out-routine.

2) If MCS gets an acknowledgement for the request, it terminates the time out routine and goes to step(5).

3) If the acknowledgement is not received in the time out perod, MCS repeats step (1) twice more.

4) In case the acknowledgement is not received after the third attempt too, MCS goes to an error routine.

5) MCS sends a byte to specify the type of communication between RTU & MCS and again enters the time out routine.

6) If MCS gets an acknowledgement for the type, it terminates the time out routine and goes to step (8) provided the type sent was other than zero. If the type is zero, then MCS goes to step (11).

7) If the acknowledgement is not received within the time out period MCS repeats step (5) twice more. If even after the third time, acknowledgement is not received,MCS goes to the error routine.

8) MCS sends the message which is kept ready in the prescribed structure (the data structure of the message is explained in the next section of this chapter).

9) MCS enters the time out routine and waits for the message received acknowledgement.

10a) If MCS gets the acknowledgement for the receipt of message, it terminates the time out routine and goes to step(12).

10b) If the acknowledgement is not received within the time out routine period MCS repeats step (9) twice more. If the acknowledgement is not received after third time also, MCS goes to the error routine.

11) MCS starts receiving the preassembled packet from RTU till end of transmission (EOT) byte is received and within the period of time out routine sends acknowledgement(ACK) or negative acknowledgement (NAK) depending upon the validity of data. If acknowledgement, it goes to step (12). If negative acknowledgement, MCS waits to again receive the same packet.(The step (11) is repeated twice more in case of NAK and after the third time it goes to error routine).

12) End of message transfer

The steps performed by MCS for this protocol is shown in Figure 5.1.

Codes;

For the transmission of data between RTU & MCS, some standard ASCII Codes are used for the data transmission which are as follows :

	ASCII Code (Hex)
SOH - Start of header	= 01
ACK - Acknowledgement	= 06

SOH - STORE OF HEADER BYTE

ACK - ACKNOWLEDGEMENT BYTE

NAK - NEGATIVE ACKNOWLEDGEMENT BYTE

EOT - END OF TRANSMISSION BYTE

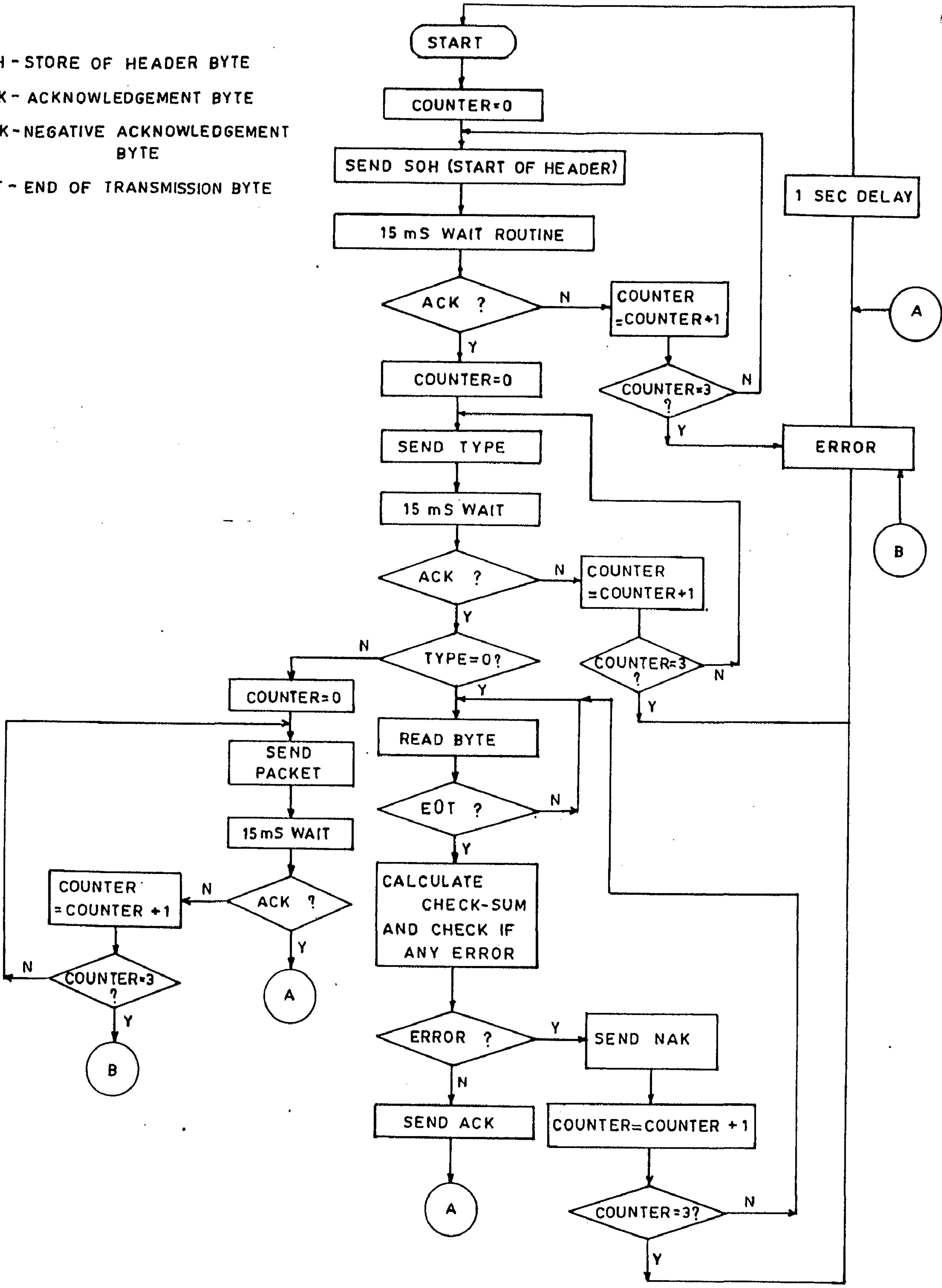


FIG. 5·1 STEPS PERFORMED BY MCS

NAK	=	Negative Acknowledgement	= 15
EOT	=	End of transmission	= 04

STEPS PERFORMED BY RTU:

When MCS interrupts RTU for data communication, following steps are performed by RTU. RTU CPU interrupts its main programme and branches to I.S.S. for communication. In this I.S.S. there are 4 entry points and according to the prior conditions CPU will branch to relevant entry point but initially the entry point=1. The concept of entry points are used to eliminate the work of CPU and to reduce the time of operation. Every time one byte comes from MCS,gives an interrupt to RTU in the form of RXReady interrpt. This RXReady interrupt is connected to RST 5-5 of RTU. RTU can perform its work irrespectable to when MCS wants to communicate. As and when MCS wants to communicate with RTU it sends byte and after reading the byte RTU with itself decide where to branch out i.e. which entry point.

(a) Entry Point 1:

1) Read byte sent by MCS and check if this byte is SOH(start of header).

2) YES - send an acknowledgement, set counter=0 and entry point =2 and return to the main program.

3) NO: Return to the main program.

(b) Entry Point 2:

1) Read type sent for data communication and check if the type is valid.

2) YES : Type sent is valid. RTU sends acknowledgement and then identifies the type sent. If type = 0, then set counter=0 and calls entry point 3 and return to the main program.

Type=Any other, then set entry step = 4 and counter=0 and return to the main program.

3) NO: Type sent is not valid. RTU increments the counter and checks if counter = 3.

YES : Entry point sets to =1 and return

NO : Entry point sets to = 2 and return.

(c) Entry Point 3:

1) RTU sends the preassembled packet and enters the time out routine of 15 ms.(and waits for ACK).

2) If ACK : Sets the entry point to 1 and returns.

3) If NAK : Increments the counter by one and checks if counter=3.

YES : sets the entry point to 1 and returns.

NO : Repeat point (1) of entry step=3.

(d) Entry Point -4:

1) Read byte until EOT & store. After checking the validity of data received send ACK or NAK.

2) If ACK : Set entry point to 1 and return

3) If NAK : Increment the counter and check whether counter=3.

YES : Entry point is set to 1 and return to the main program.

NO : Entry point is set to 4 and return to the main program.

Points performed by RTU are given in Fig.5.2.

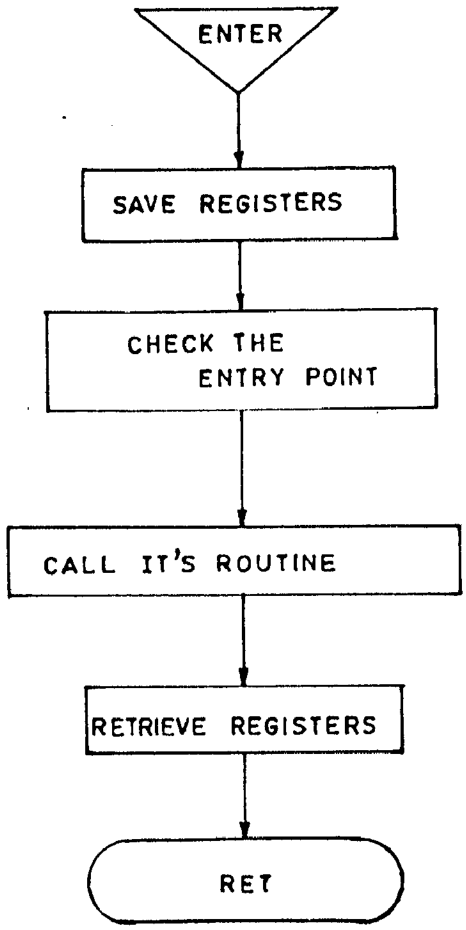


FIG. 5·2 (a) I.S·S FOR RST 5·5 THIS PROGRAM CALLS DIFFERENT ENTRY POINTS

(a) ENTRY POINT-1

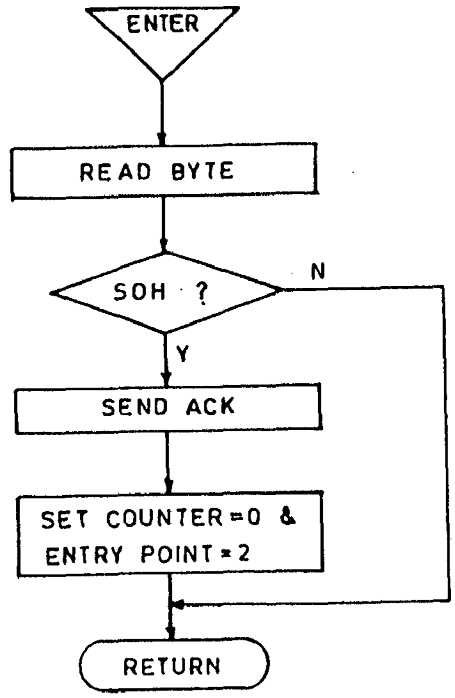


FIG. 5·2·1(b) READ SOH BYTE

(b) ENTRY POINT-2

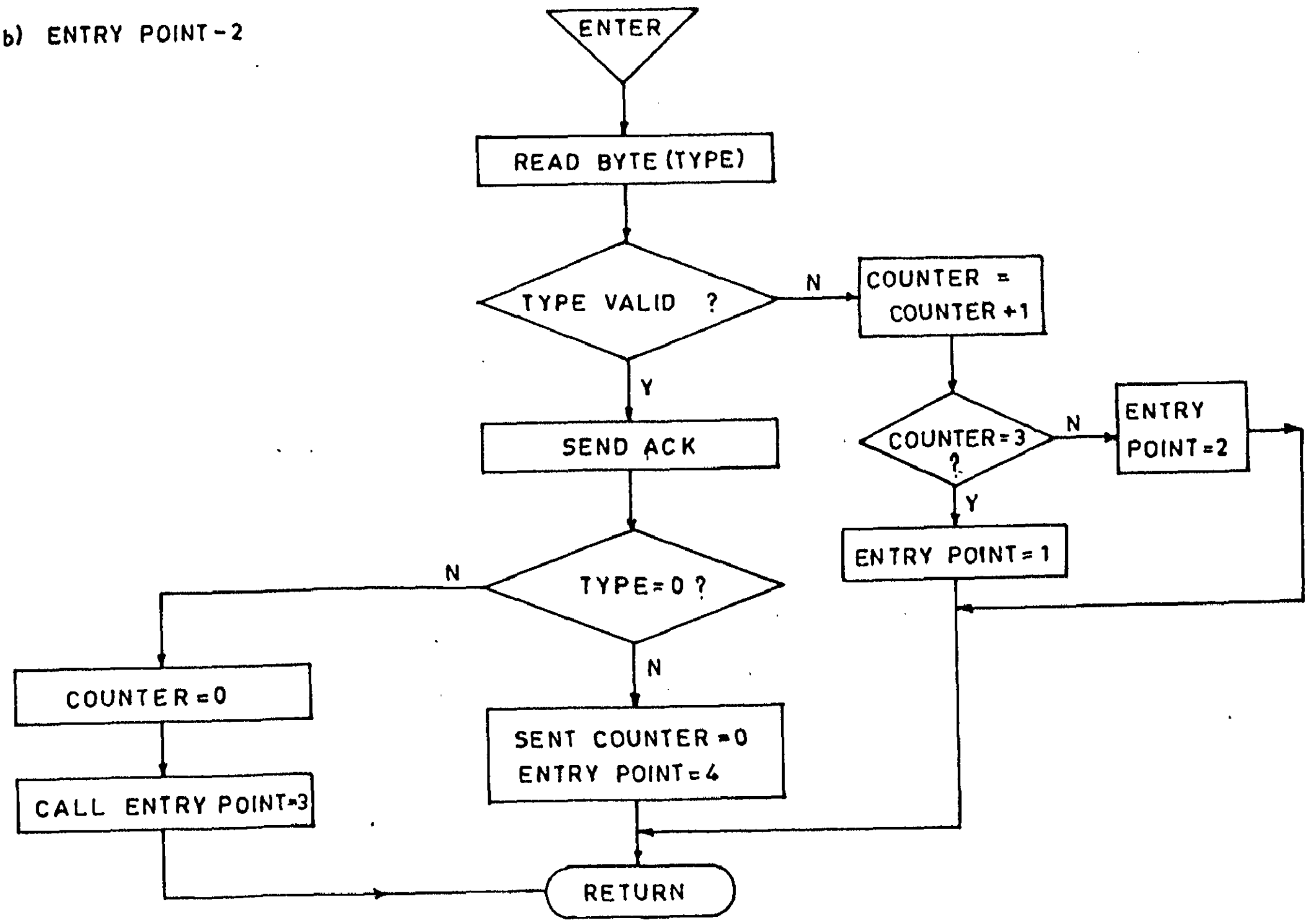


FIG. 5·2·2 (b) READ TYPE BYTE FOR COMMUNICATION

(c) ENTRY POINT-3

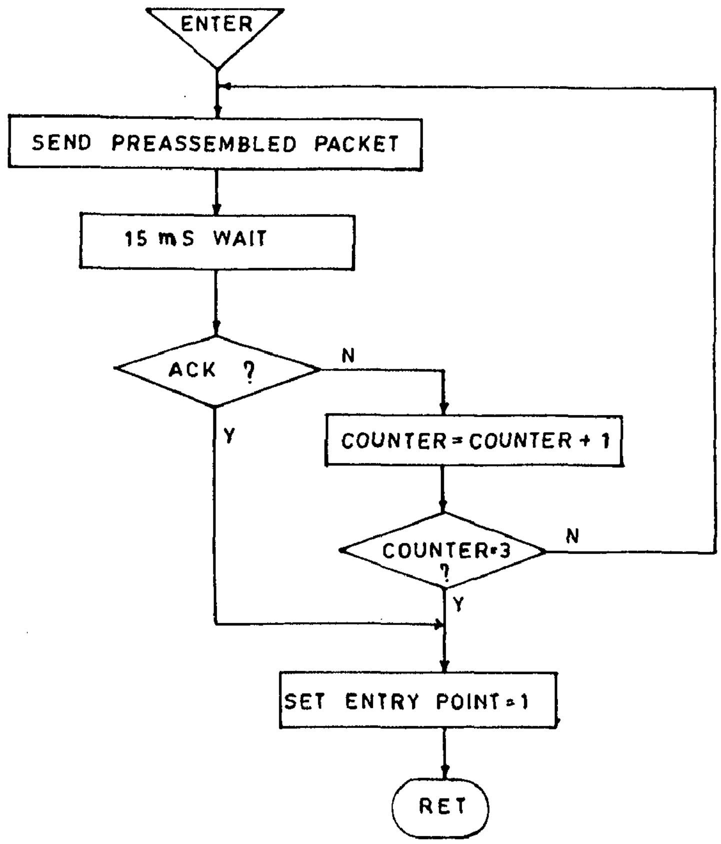


FIG. 5·2·3 (b SEND PACKET TO MCS

(d) ENTRY POINT-4

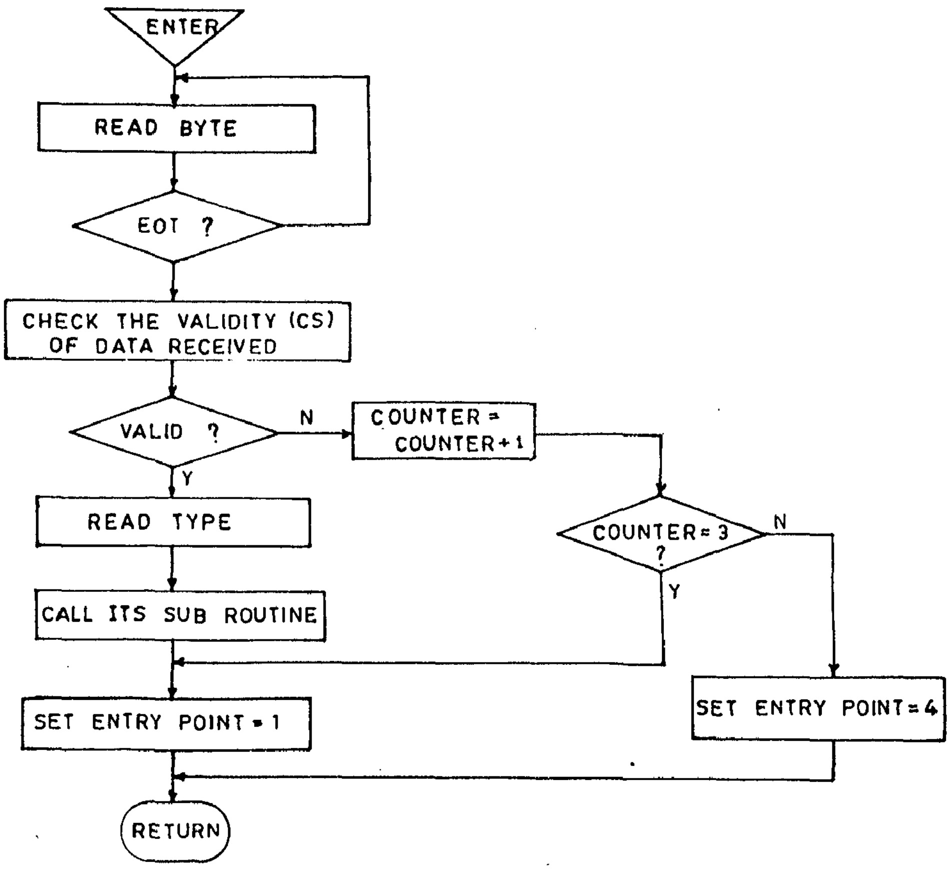


FIG. 5·2·4 (b) RECEIVE PACKET FROM MCS

FIG. 5·2 (b) DETAILS OF DIFFERENT ENTRY POINTS

5.3 DATA STRUCTURE:

All the messages which fly between MCS & RTU should be put in the format shown in Figure 5.3. It is a simple data structure with one byte message header.

The communication between the two stations is always in ASCII Code. The STX, ETX & EOT all the three have standard ASCII codes. STX and ETX are used so that the start and end of data communication are indicated. The number of data bytes is dependent upon the type of communication between RTU and MCS. The one byte of data is always split into two bytes for transmission between the two stations. For example if the byte 2A has to be transmitted then this byte will be transmitted as 32 and 41 between RTU and MCS. This implies that number of data bytes for transmission are just the half the number of bytes that are transmitted. For this reason checksum is also transmitted in two bytes although the checksum itself is a single byte data.

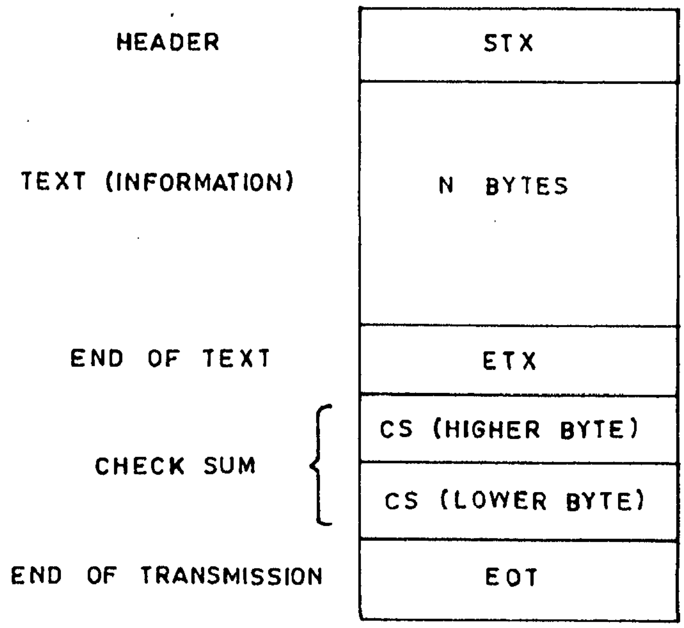
Between the ETX and EOT, Checksum is sent to check the validity of received data. At the end EOT byte is sent to indicate the end of transmission between two stations.

5.4 RTU SOFTWARE FOR COMMUNICATION:

Following subroutines and main program comprise the RTU software:

a) <u>HEX TO ASCII CONVERSION:</u>

(HXTA) - This subroutine converts a block of hex data stored in memory to its ASCII equivalent and stores it the latter in other memory area sequentially as shown in Fig.5.4.2. One byte



	ASCII CODE (HEX)
STX - START OF TEXT	02
ETX - END OF TEXT	03
EOT - END OF TRANSMISSION	04

FIG. 5·3 DATA STRUCTURE

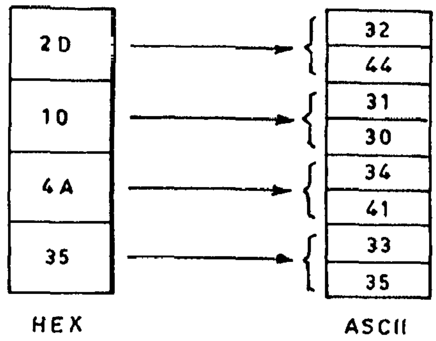


FIG.5·4·1 EXAMPLE ON HEX TO ASCII CONVERSION

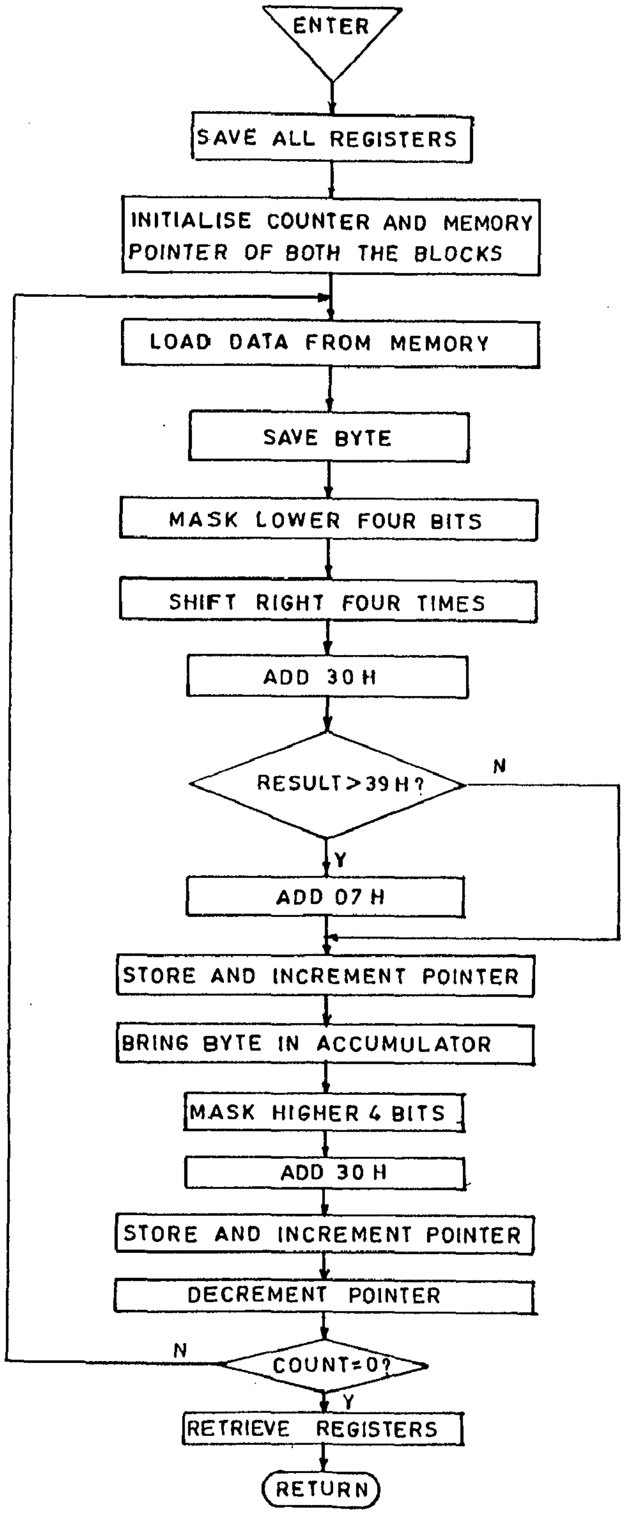


FIG.5·4·2 LOGIC TO CONVERT HEX TO ASCII

of hex data is split into two bytes of ASCII. An example is shown in Fig.5.4.1.

From the example of Fig.5.4.1, ASCII it is clear that memory requirement after conversion is just double.

b) CALCULATION OF CHECKSUM:

Checksum is the one's compliment of the sum of all the data bytes. The sum is done by discarding the carry and it is an 8-bit number. The ASCII equivalent of checksum is a two byte data.This routine calculates checksum of data stored in a block of memory and stores and then converts it into its ASCII format and again store so that it is really available for communication. The flow chart is shown in Fig.5.5.

c) ASCII TO HEX CONVERSION (ASTH):

The routine converts a block of ASCII data collected from MCS to its HEX equivalent. The logic is shown in Fig.5.6.1, 5.6.2.

d) TRANSFERS THE BLOCK OF DATA TO MCS: (DATA TRANSFER):

This routine adds STX, ETX & EOT to the block of ASCII Data stored in memory. The flow chart is shown in Fig.5.7.2. and example shown in Fig.5.7.1.

This block of Fig.5.7.1 is transfer first by adding STX(02) then data (30,35) after that ETX(03) and then CS(41,46) and then EOT (04).

02,30,35,03,41,46, 01

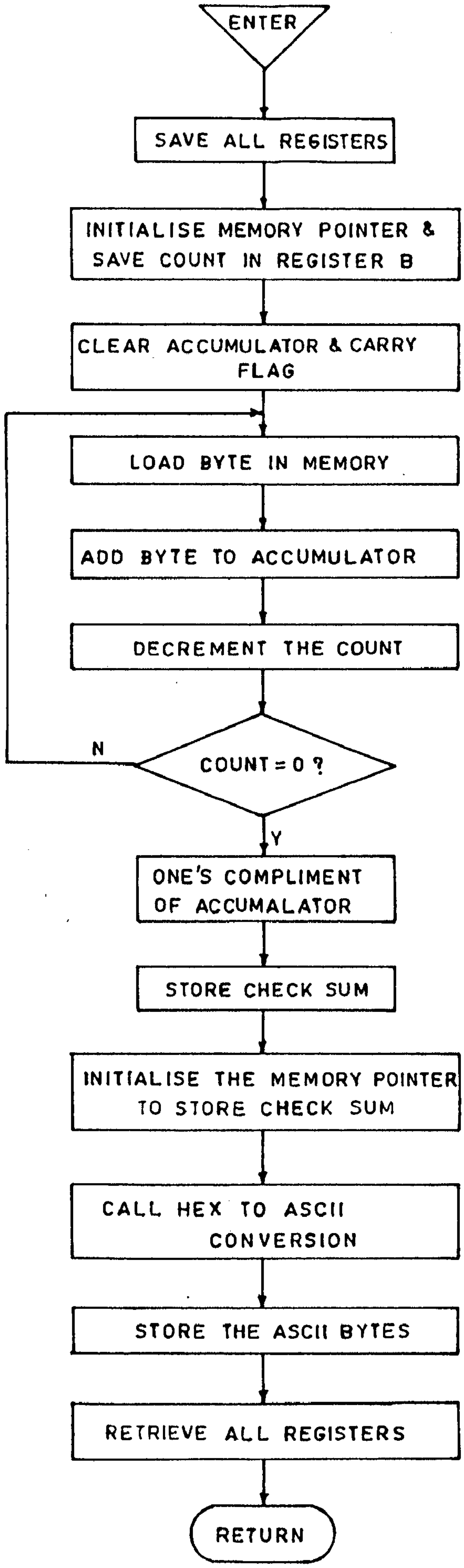


FIG. 5·5 CALCULATION ON CHECK SUM

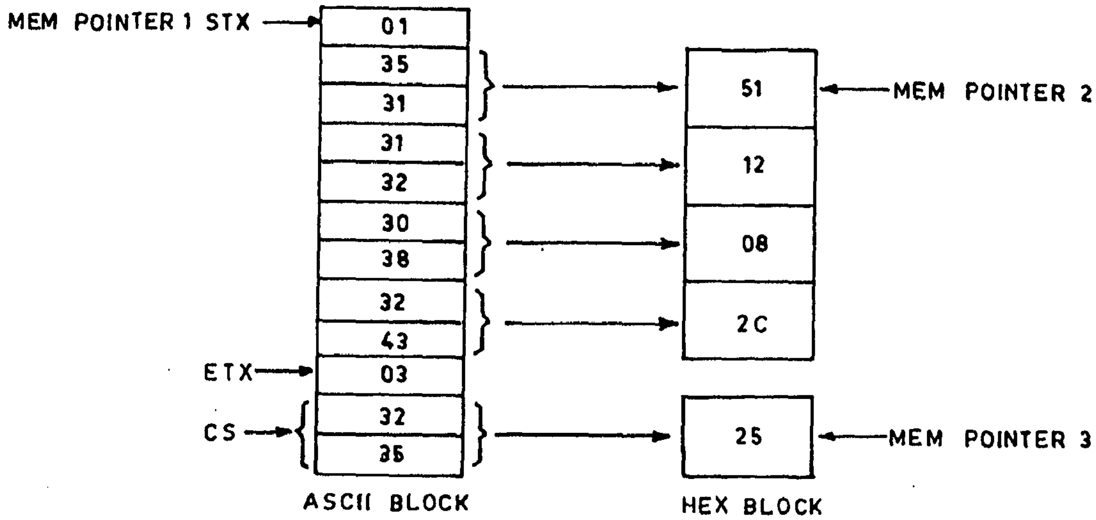
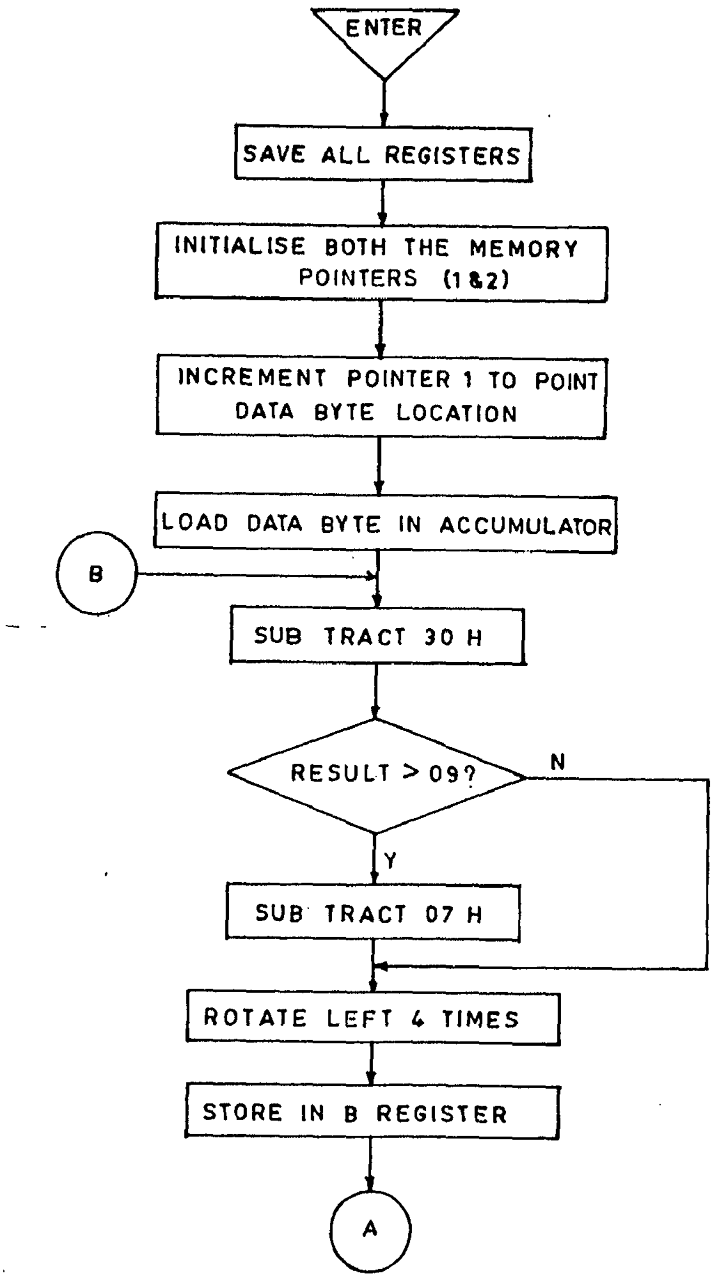
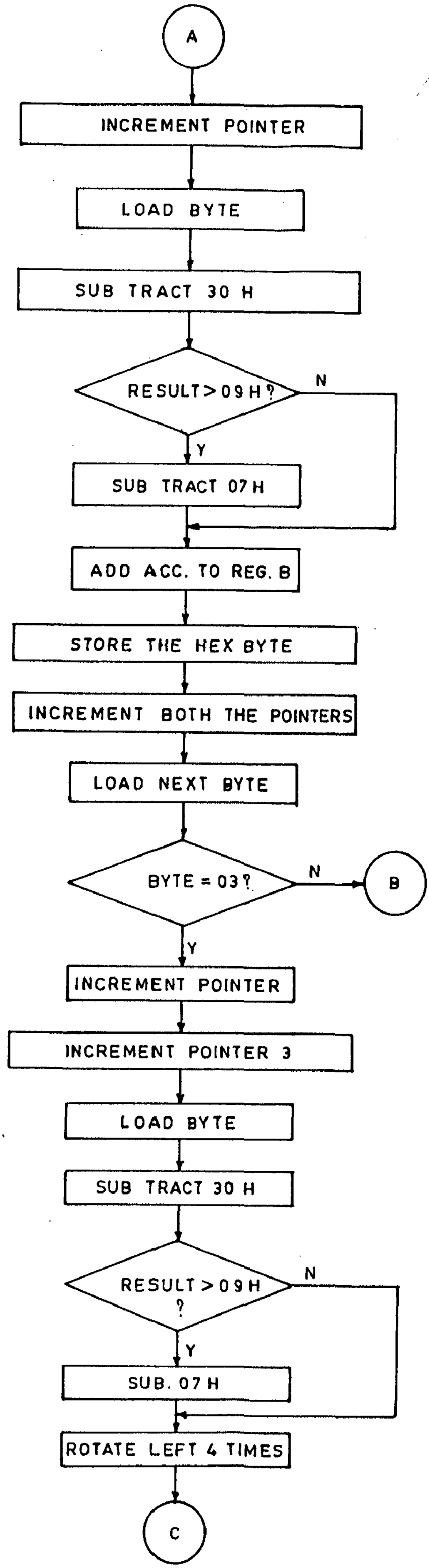


FIG. 5·6·1 EXAMPLE TO CONVERT ASCII BLOCK TO HEX BLOCK




A
INCREMENT POINTER
LOAD BYTE
SUB TRACT 30 H
RESULT > 09 H ?
N
Y
SUB TRACT 07 H
ADD ACC. TO REG. B
STORE THE HEX BYTE
INCREMENT BOTH THE POINTERS
LOAD NEXT BYTE
BYTE = 03 ?
N
B
Y
INCREMENT POINTER
INCREMENT POINTER 3
LOAD BYTE
SUB TRACT 30 H
RESULT > 09 H ?
N
Y
SUB. 07 H
ROTATE LEFT 4 TIMES
C

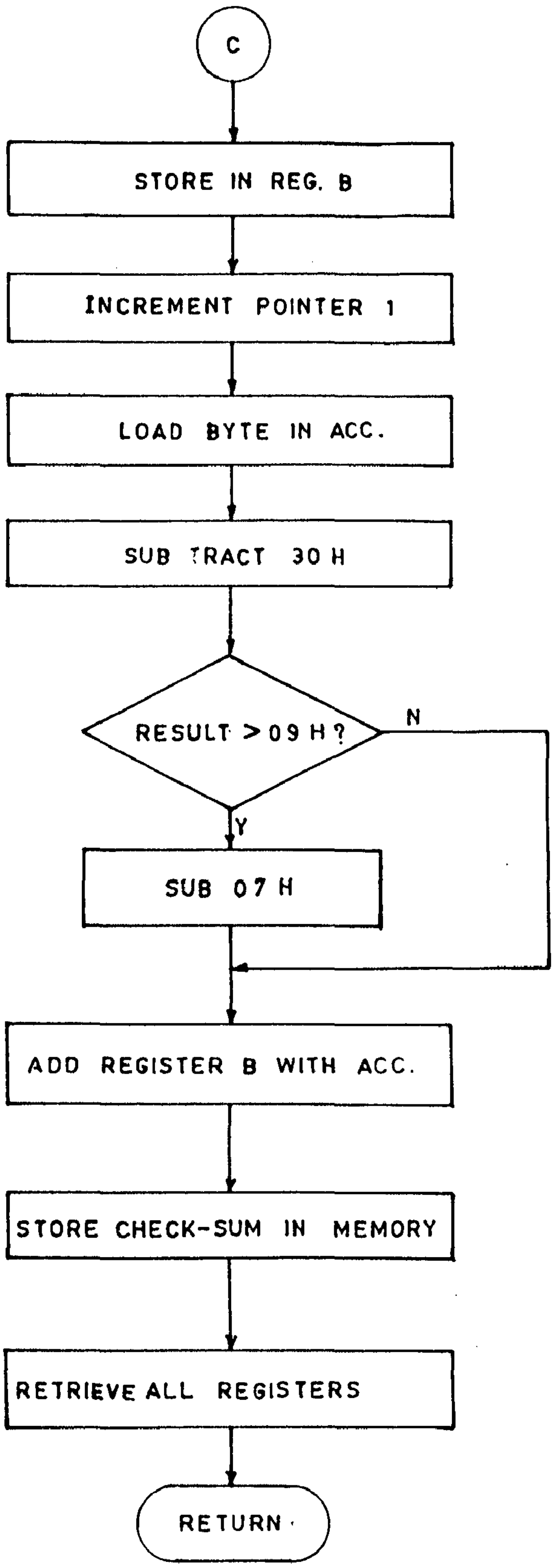


FIG. 5·6·2 ASCII TO HEX CONVERSION

FIG. 5·7·1 DATA AND CHECK SUM EXAMPLE TO BE TRANSFERED

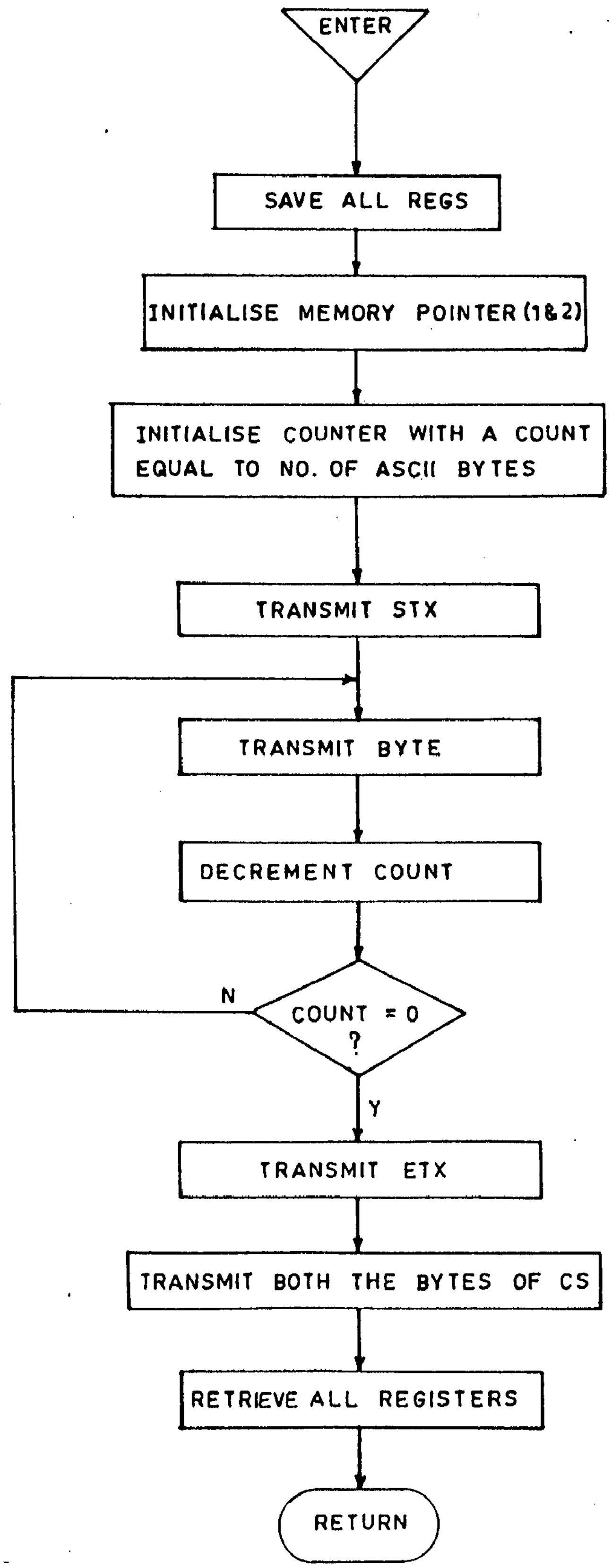


FIG. 5·7·2 SEND PACKET

Details of subroutine and flowcharts for all the four entry points are given earlier.

There are different type of data communication according to the type sent by MCS, RTU will respond. The details of different types are as follows:

Type:

0 - for sending all the data from RTU to MCS

1 - Start Plant

2 - Stop plant

3 - Start device

4 - Stop device

The number of the device or plant to be started or stopped is sent by MCS only.

5 - Renew reference value of variable (given by MCS)

6 - Change the higher limit of variable

7 - Change the lower limit of variable

8 - Reset counter 0

9 - Reset counter 1

A - Read counter 0

B - Read counter 1

e) DISPLAY THE COMMAND (SUB31 TO SUB 34):

The logic used for these four subroutines are same and their flow chart is shown in Fig.5.8. This subroutine displays the command start or stop the plant or start or stop the switching device. The number of plant or switching device is loaded in MEM17

f) RENEW REFERENCE VALUE (SUB 35):

This subroutine is for type-5 of control command. This will renew the reference value of the variable. The variable number is sent by MCS itself. The flow chart is shown in Fig.5.9.

g) CHANGE HIGHER OR LOWER UNIT (SUB 36 AND SUB 37):

These two routine changes the higher or lower limit of variable whose value is given by MCS.

h) RESET COUNTER (SUB 38 AND SUB 39):

These routine reset the counter 0 or counter 1 to FFFFH.

i) READ COUNTER (SUB 41 AND SUB 42):

These routine reads the value of counter 0 or counter 1 from RTU.

j) I.S.S. OF RST 5.5:

In this interrupt service routine the entry point is checked by the number stored in memory Entry and accordingly the entry point routine is called. After performing these functions the routine returns back to main program.

5.5 MCS SOFTWARE FOR COMMUNICATION:

(a) Main Programme:

Main program for communication is always loaded from 100H and code & data segment are initialised in one segment.

In main program following functions are performed:

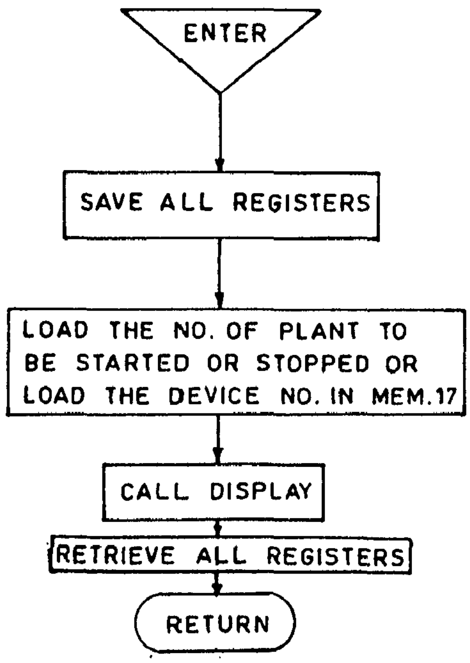


FIG. 5·8 DISPLAY ROUTINES (SUB 31 TO SUB 34)

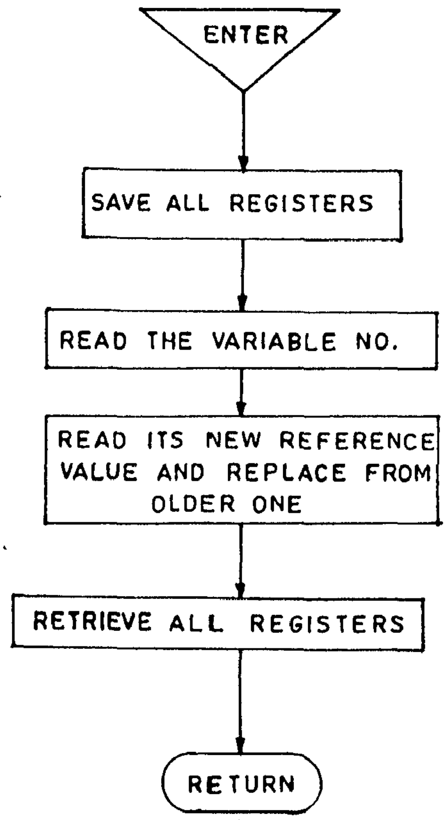


FIG. 5·9 RENEW REFERENCE VALUE (SUB 35)

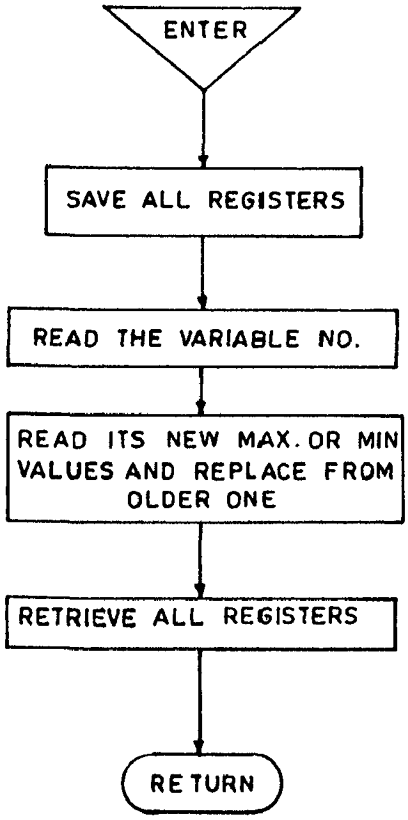


FIG. 5·10 CHANGE HIGHER OR LOWER LIMIT
(SUB 36 OR SUB 37)

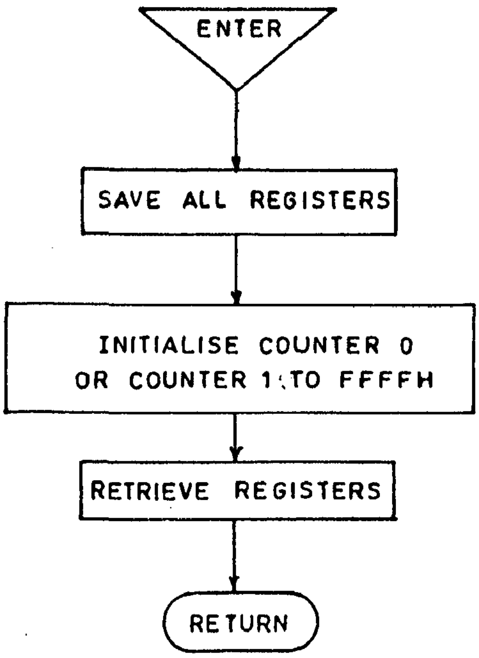


FIG. 5·11 INITIALISE COUNTER 0 OR COUNTER 1
(SUB 38 OR SUB 39)

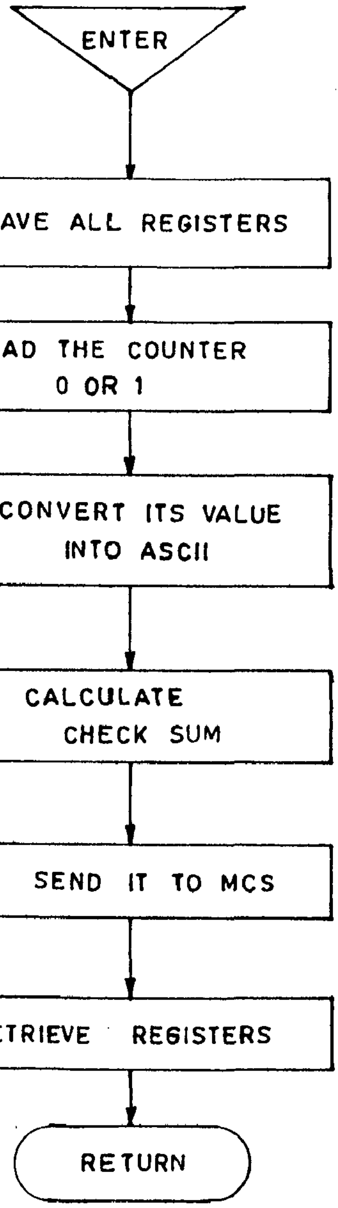


FIG. 5·12 READ COUNTER 0 OR COUNTER 1 AND TRANSMIT TO MCS

1) Gives an interrupt to RTU by sending SOH at an interval of 1 second.

2) After sending the SOH it reads keyboard to input the type of data transmission. If the type is beyond the range of (30H to 42 H) it displays the error in the form of a beat. And if the type is valid then it performs its function. the flow chart is shown in Fig.5.1.

b) Initialisation of Communication Port(Init):

The subroutine initiales the communication port for 4800 bands, even parity, 1 stop bit and 8 bits per character. The procedure is shown in Fig.5.1.3.

c) Send byte to communication port:(send-byte):

This subroutine shown in Fig.5.14 sends the byte in register Al to communication port by using INT14. This function is repeated thrice in case of any error.

d) Send the whole preassembled packet(send-pkt):

In the routine shown in Fig.5.15 the whole packet which is preassembled earlier is sent to RTU with the help of routine send-byte. In this procedure, the offset of transmitter buffer is loaded in SI(Source Index) register that will final give the EA (effective address) and the number of bytes to be transferred is loaded in CX register.

e) Receive byte from RTU (get-byte):

This routine shown in Fig.5.16 receives a byte in AL sent on communication port and checks for any error. In case of error that byte is read thrice with the help of INT 14H.

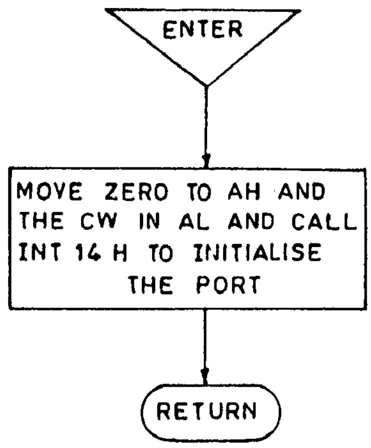


FIG. 5·13 INITIALISATION OF COM-PORT

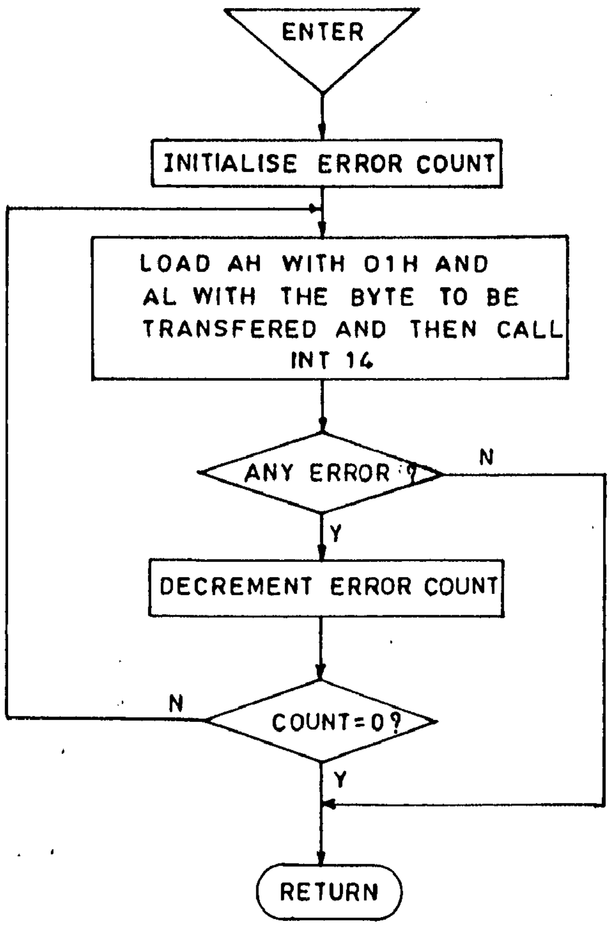


FIG. 5·14 SEND BYTE TO COM-PORT

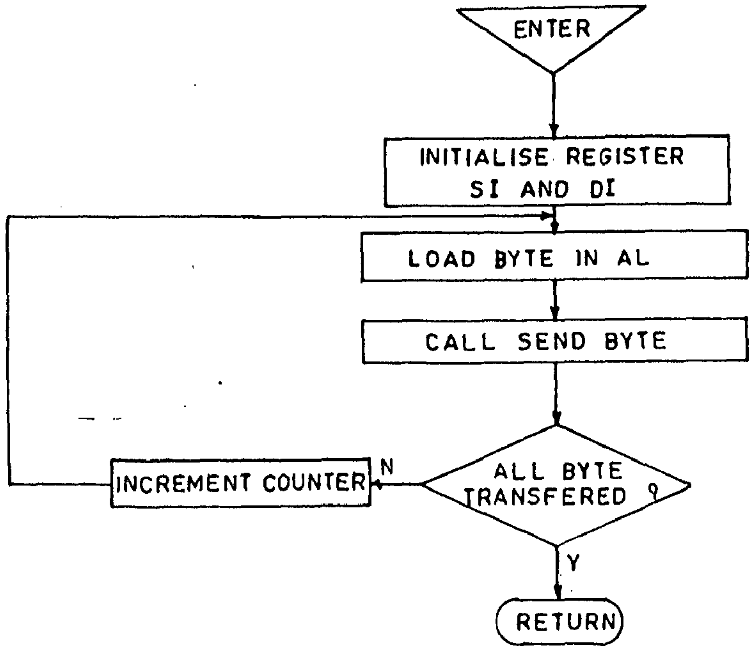


FIG. 5·15 SEND PACKET

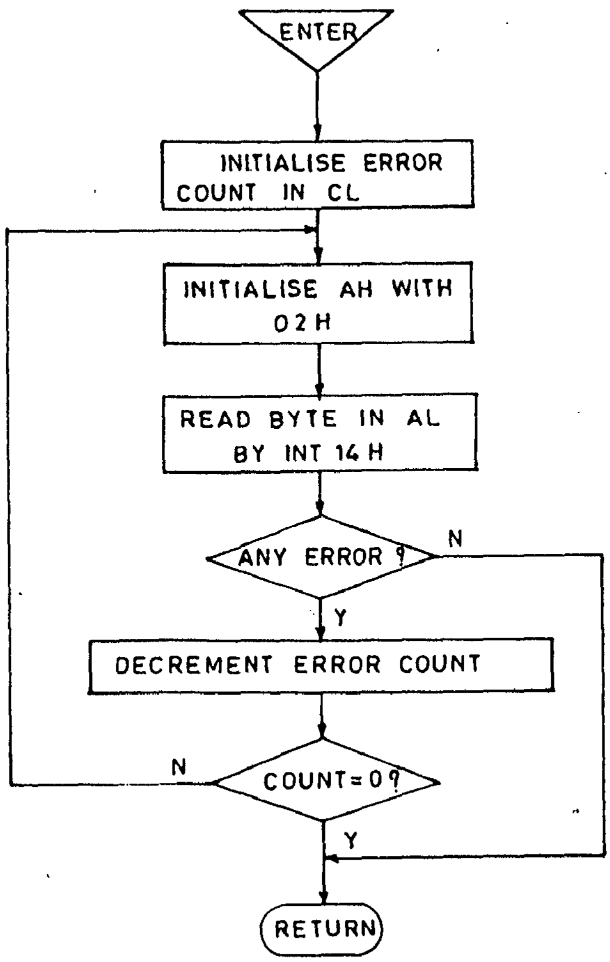


FIG. 5·16 GET BYTE FROM COM-PORT

f) Receive packet from RTU (GET PKT):

This routine (shown in Fig.5.17) receive packet from RTU till EOT and store the packet in RX-buffer.

g) Make packet of number of HEX Data byte(make-pkt):

This subroutine (shown in Fig.5.18) converts the Hex data bytes to equivalent-ASCII bytes and then adds STX, ETX & EOT to ASCII data. With in this routine checksum is also calculated and added between ETX & EOT. Before calling this routine CX should contain number of hex data bytes.

h) Convert HEX byte to Numeric ASCII Word(HEX-to-ASCIII):

This routine (shown in Fig.5.19) converts hex byte to ASCII word. This should be called with the Hex byte in Al register. And this routine returns with word in Ax register.

i) Convert - ASCII word to Hex byte(ASCII-to-HEX):

This subroutine (shown in Fig.5.20) converts the ASCII word in register AX to HEX byte. This routine return with byte in AL register.

j) Calculate checksum(CHK-SUM):

This routine(shown in Fig.5.21) calculates the check-sum and returns with checksum in AL.

k) Send the whole packet to RTU(send-out):

This routine (shown in Fig.5.22) first makes the packet from HEX data and then sends the packet to RTU. Check if ACK is received or not. In case acknowledgement is not received within 15 ms then the packet is sent twice and if after sending the packet three times acknowledgement is not received then goes to error routine. Before entering the routine ex should be loaded with the number of HEX bytes.

1) <u>Receive the whole packet from RTU(GET-IN):</u>

This routine (shown in Fig.5.23) receive the whole packet from RTU and then checks if the packet is received correctly by checking checksum sent by RTU to the checksum calculated by MCS. If the packet is received properly without any error, acknowledgement is sent to RTU else, NAK acknowledgement is sent to RTU and accordingly Nak-error thus is set.

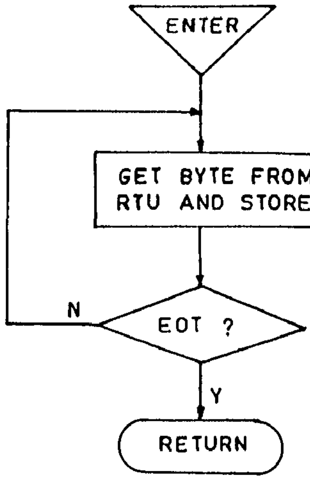


FIG. 5·17 RECEIVE PACKET

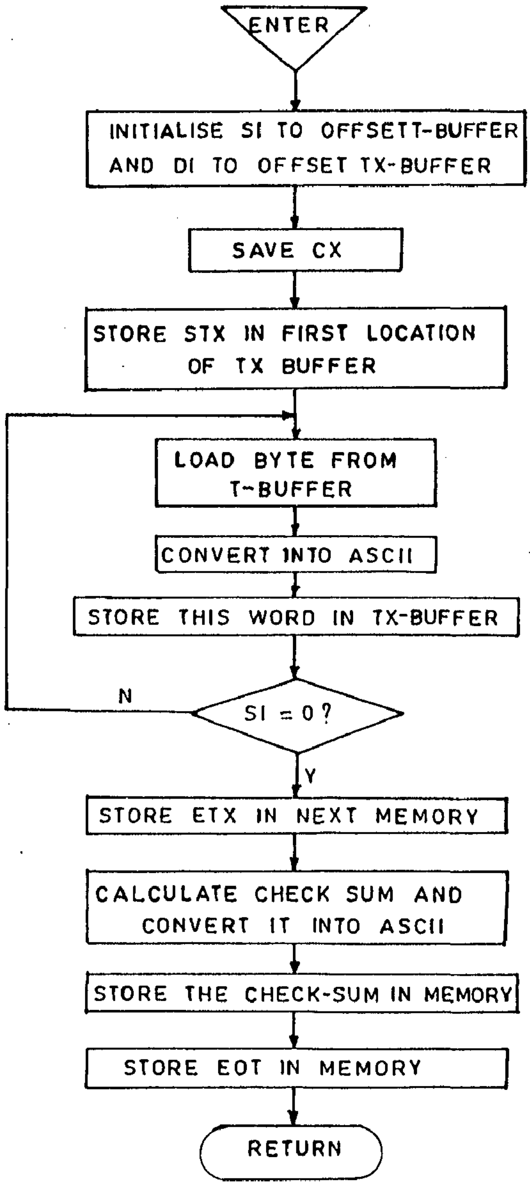


FIG. 5·18 MAKE PACKET

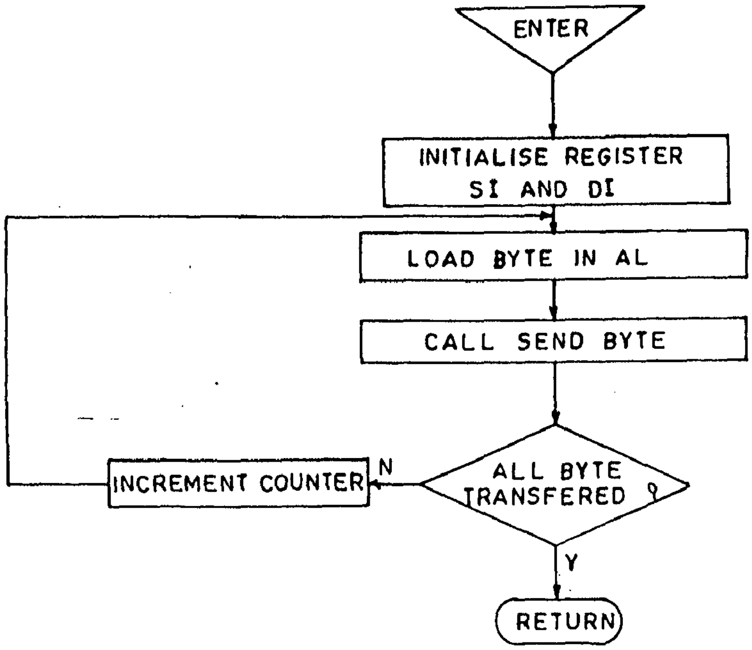


FIG. 5·15 SEND PACKET

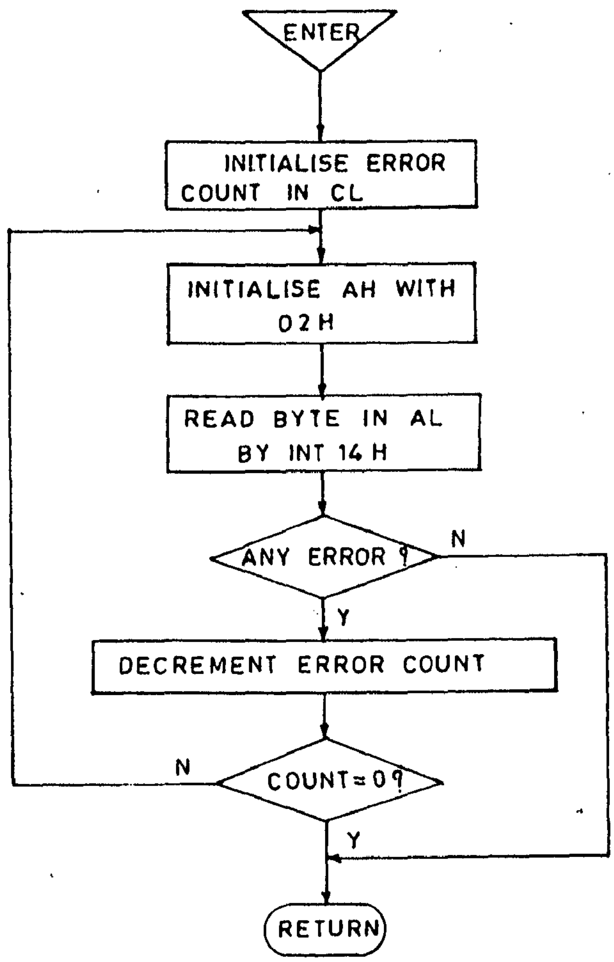


FIG. 5·16 GET BYTE FROM COM-PORT

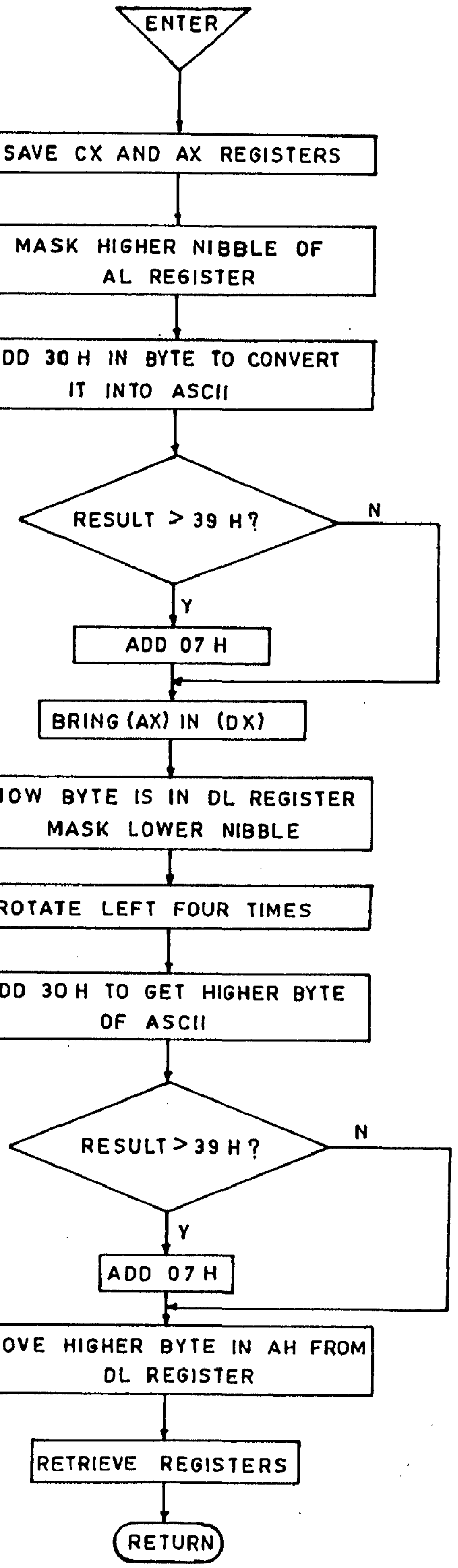


FIG. 5.19 HEX BYTE TO ASCII WORD CONVERSION

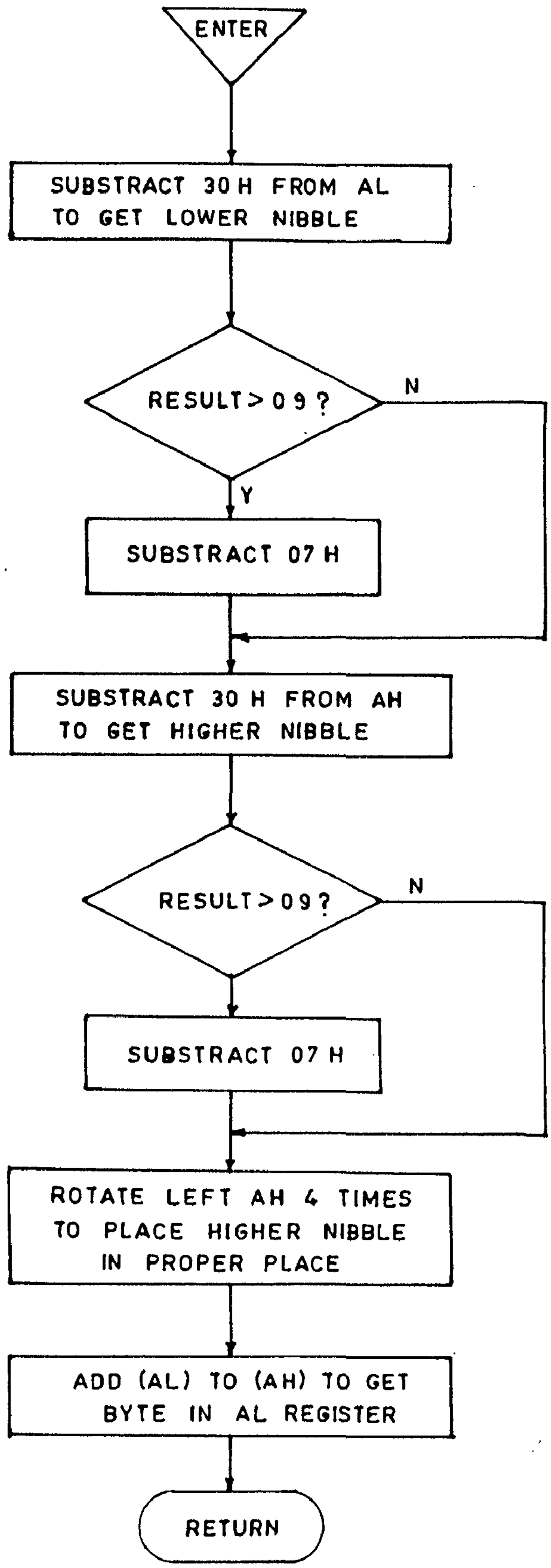


FIG. 5·20 ASCII WORD TO HEX BYTE CONVERSION

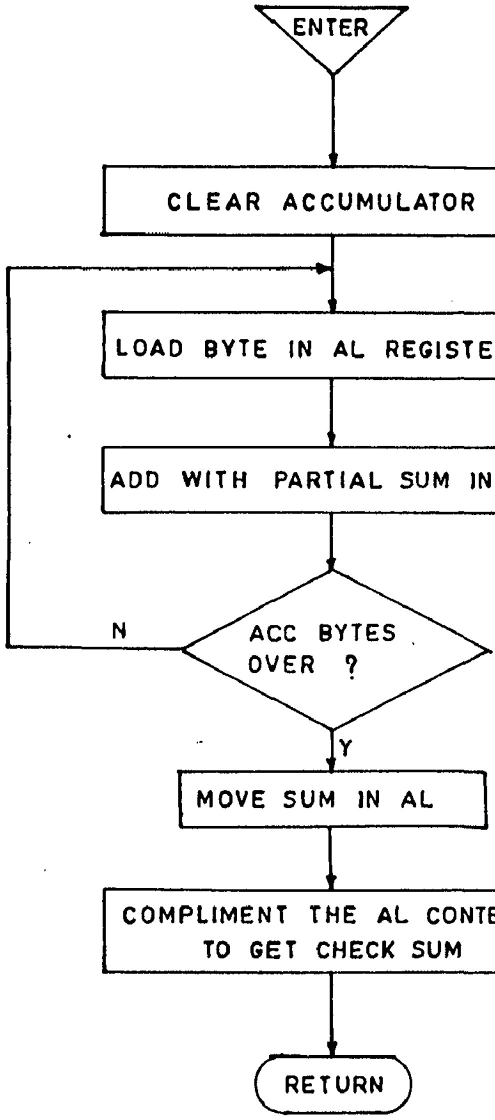
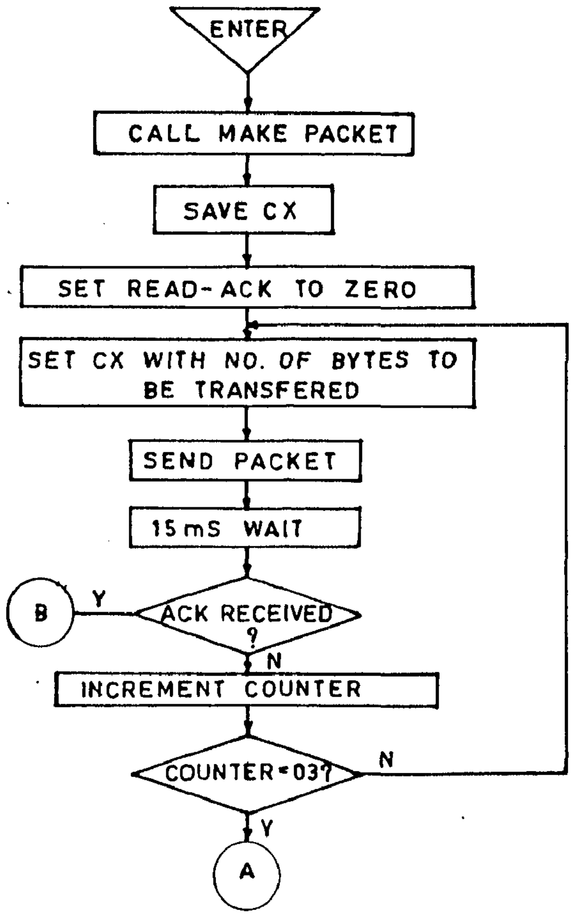


FIG. 5·21 CALCULATION OF CHECK-SUM



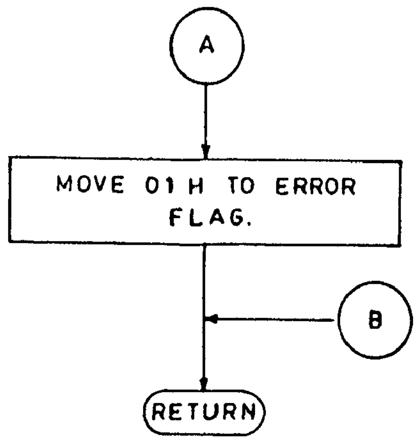


FIG. 5·22 SENDS PACKET

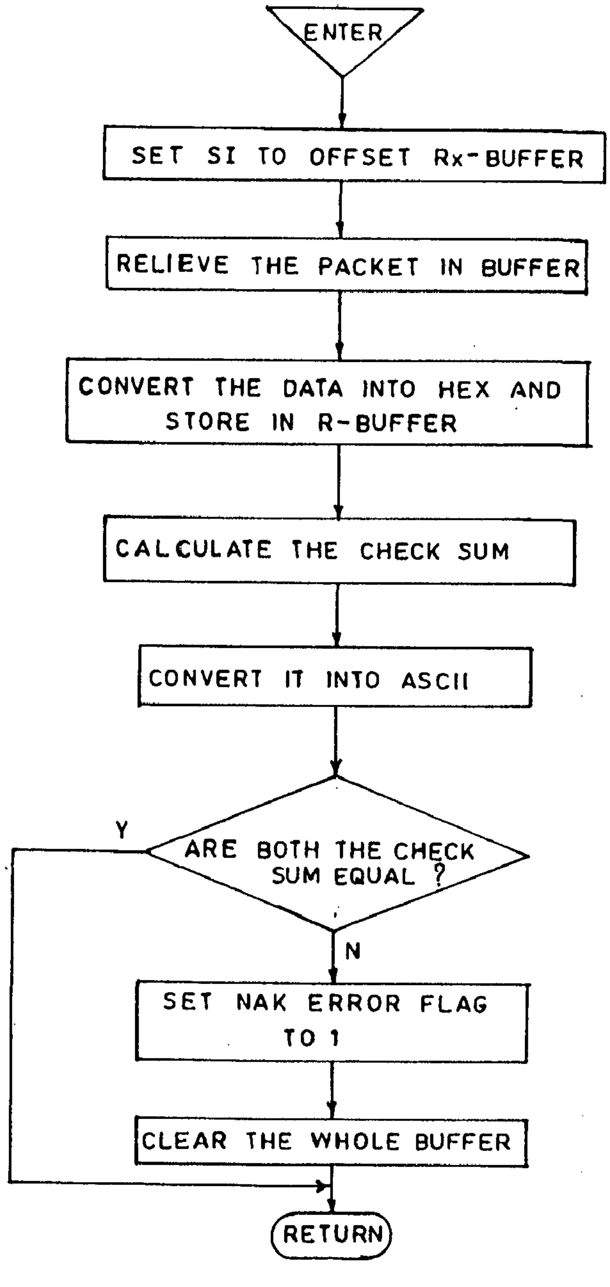


FIG. 5·23 GET-IN THE PACKET

CHAPTER-6
CONCLUSIONS & SCOPE OF FURTHER WORK

6.1 CONCLUSIONS

A review of the developments in remote control, telemetry and supervisory control techniques since the World War II has been presented. Trends in communication with respect to the impact of the development of transistor, large automatic tracking antennas, phase-locked FM detector and technologically superior communication mediams, are also seen in detail. The use of computer in SCADA is overviewed. Based on this knowledge and the knowledge of computers a "Two level SCADA System" with one master control station (MCS-80286 PC based) and one remote terminal units (RTU-8085 uc based) has been developed and implemented.

Some important facilities necessarily required at the remote terminal units that are achieved by using the hardware modules of micro computer are the inputting, display and monitoring of analog & digital variables, PID control, digital control and integration of pulse inputs.

Software routines and programs has been developed for data acquisition, processing of the data, displaying the relevant information on CRT and transmitting it to the master control station on demand.

Facilities at the master control station are broadly the control of variable, display of received information and communication with remote terminal units.

The software for these facilities has been developed and implemented. Display software is in Fortran-77, and communication and control software is in 8086 assembly langauge.

Communication between RTU and MCS is serial using RS232C link. A protocol for exchange of information between the two stations for minimum error during communication has been designed and implemented.

6.2 SCOPE FOR FURTHER WORK:

The system can be explained using additional RTU's by providing additional communication ports in the hardware of the MCS

By adding sufficient processing capabilities with physically distributed RTU's. This system can be developed into a powerful computer network with the help of a proper networking software. The configuration for networking can be either a ring or a star.

Modems can be used where different RTU's are placed at physically very large distances. To obtain high communication efficiency, optical fibre can be used as the communication medium. This can improve communication speed and almost eliminate data loss.

To enhance user RTU interface in the system, graphic software can be used to display system configuration and its relevant information in the form of mimic diagrams. Colour graphic display can be done on MCS to improve readability of the mimic diagram.

To enhance facilities on RTU, processing capabilities of the CPU and RTU can be increased by using 16 bit processor. Fast speed, high throughput and better real time interface can thus be obtained. Similarly a 32-bit system for MCS can be used.

A 12-bit instead of 10-bit ADC can be used in the RTU to provide better resolution, reduce the quantisation error and improve sensitivity.

From the MCS software one of the more popular and control oriented language like C can be used in place of assembly language. This will improve maintainability of software and hardware, improve portability of software and most importantly facilitate changes in the software.

Real time system like iRMS is used to increase the throughput.

The SCADA system can be expanded into a powerful management information system using on appropriate Data Base Management package like ORACLE, INGRESS, FOCUS and UNIFY. Different statistics about the performance, maintenance etc. of all RTU's could be captured into the data base and statistical reports for a RTU could be taken out at a later stage for any period of operation.

REFERENCES

1. ELLIOT L. GRUENBERG (1967) : Fundamentals in 'Hand book of Telemetry and Remote Control' (Page 1-2 - 1-4)
2. H. RAMESH, S.GHOSH, M.G. DANIEL (1988): Systems Engineering Approach : Relevance to SCADA in 'All India Seminar on SCADA for Power Systems and Industries' (Page 11-4-1-11-4-11)
3. SURENDER KUMAR (1988) : Trends in SCADA Systems in 'All India Seminar on SCADA for Power Systems and Industries'(page 1-1-5)
4. SURESH KAMATH, S.SUNDARARAJAN, SATISH MOKKAPATI: An Optimal Solution to SRC/DAS for 25 KV Traction Supply in 'All India Seminar on SCADA for Power Systems and Industries' Page (11-6-3-11-6-4)
5. J.G. TRUXAL, M.L. SHOOMAN, W.B. BLESSER, J.W. CLARK: Remote Control in 'Hand book of Telemetry and Remote Control' Page (15-2-15-3).
6. C.H. HOEPPNER, C.H. DOERSAM, J.H. SMITH, J.F.BRINSTER, HANS SCHARIA-NIELSEN, LAVEAGNE E-WILLIAMS :-RADIO-TELEMETRY SYSTEMS in 'Handbook of Telemetry and Remote Control' Page (4-2-4-3,4-24-4-25).
7. GUNTHER SWOBOOA: The Design of Telecontrol Installations in 'Telecontrol : Mwethods and applications of Telemetering & Remote Control' Page (312-314).
8. SUBHASH.C. CHOPRA, RAVI RAINA: Supervisory Control and Data Acquisition System - An Overview in 'All India Seminar on SCADA for Power Systems and Industries' Page (111-4-4-111-4-5).
9. LAMBA, Y.P.SINGH(1989): Architectures for Computer Control Systems in 'National Workshop on Distributed Computer Control Systems' Page (1-3).
10. C.N. VASUDEVAN(1989): Advanced Control and Optimisation in Refineries in 'National Workshop on Distributed Computer Control Systems' page (1-8_.
11. ARMBRUST & FORGERON (1988) : BIOS Interrupts and Function Request in 'Programmers Reference Manual for IBM Personal Computers' page (504-707).

APPENDICES

APPENDIX - A

```
        ORG     49152
;define external and public variables
        EXTRN   MRMS1,MAVRAGØ,MMIN1,MMAX1
        PUBLIC  ADCREAD,DISP4,DISP1,DISP2,DISP3
        EXTRN   MEMMIN,MEM94C,MAXMAX,BYTEMAX,BYTEST,MINMIN,BYTEMIN,BYTESTØ,MEM1
        PUBLIC  RMS,MULT16,DIV,ADD32,AVRAG,BINTA,SQRT,MAX,MIN,IRØ,IR1,IR2,STATU
        EXTRN   MEM1ØØ,MEM1Ø1,MEM1Ø2,MEM1Ø3,MEM1Ø4,MEM1Ø5,MEM1Ø6,MEM1Ø7,MEM4
        EXTRN   TEMP1,MEM9Ø,MEM92,MEM88,MEM86,MEM95,TEMP2,MEM94A,MEMMAX,MEM94B
        EXTRN   MADD32,MMULT16,MEM6,MDIV16,MEMA,MPOINT,TEMP4,MBINTA,TEMPMUL

;This routine calculates the RMS value of N numbers (1Ø-bit) stored
;in sequential memory location.

RMS     PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        LXI     H,MEM1ØØ        ;initialise the memory pointer of addition rout
        SHLD    MADD32          ;store in MADD32
        LXI     H,MEM92         ;initialise the memory pointer for divident men
        SHLD    MEMA            ;and store in MEMA
        LXI     H,TEMPMUL       ;initialise the pointers for multiplication &
        SHLD    MEM6            ;store in memory mult16 & mem6
        SHLD    MMULT16
        MVI     A,1ØH           ;load count in accumalator
        LHLD    MEM95
        MOV     C,L
        MOV     B,H             ;save the starting address of block in BC
        LXI     D,MEM88         ;multiplicant memory in in DE
        LXI     H,MEM86         ;multiplier memory in HL
LOOP4   STA     TEMP2           ;save the count
        LDAX    B
        MOV     M,A
        XCHG
        MOV     M,A
        INX     H
        INX     B
        INX     D
        LDAX    B
        MOV     M,A
        XCHG                    ;set multiplier & multiplicant with the
        MOV     M,A             ;number to be squared
        CALL    MULT16          ;call mult16 (for 16 by 16 Bit multiplication)
        DCX     D
        DCX     H               ;decrement D,H pointer
        INX     B
```

```
ORG     49152
;define external and public variables
EXTRN   MRMS1,MAVRAGØ,MMIN1,MMAX1
PUBLIC  ADCREAD,DISP4,DISP1,DISP2,DISP3
EXTRN   MEMMIN,MEM94C,MAXMAX,BYTEMAX,BYTEST,MINMIN,BYTEMIN,BYTESTØ,MEM1
PUBLIC  RMS,MULT16,DIV,ADD32,AVRAG,BINTA,SQRT,MAX,MIN,IRØ,IR1,IR2,STATU
EXTRN   MEM100,MEM101,MEM102,MEM103,MEM104,MEM105,MEM106,MEM107,MEM4
EXTRN   TEMP1,MEM90,MEM92,MEM88,MEM86,MEM95,TEMP2,MEM94A,MEMMAX,MEM94B
EXTRN   MADD32,MMULT16,MEM6,MDIV16,MEMA,MPOINT,TEMP4,MBINTA,TEMPMUL

;This routine calculates the RMS value of N numbers (10-bit) stored
;in sequential memory location.

RMS     PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        LXI     H,MEM100        ;initialise the memory pointer of addition rout
        SHLD    MADD32          ;store in MADD32
        LXI     H,MEM92         ;initialise the memory pointer for divident men
        SHLD    MEMA            ;and store in MEMA
        LXI     H,TEMPMUL       ;initialise the pointers for multiplication &
        SHLD    MEM6            ;store in memory mult16 & mem6
        SHLD    MMULT16
        MVI     A,10H           ;load count in accumalator
        LHLD    MEM95
        MOV     C,L
        MOV     B,H             ;save the starting address of block in BC
        LXI     D,MEM88         ;multiplicant memory in in DE
        LXI     H,MEM86         ;multiplier memory in HL
LOOP4   STA     TEMP2           ;save the count
        LDAX    B
        MOV     M,A
        XCHG
        MOV     M,A
        INX     H
        INX     B
        INX     D
        LDAX    B
        MOV     M,A
        XCHG                    ;set multiplier & multiplicant with the
        MOV     M,A             ;number to be squared
        CALL    MULT16          ;call mult16 (for 16 by 16 Bit multiplication)
        DCX     D
        DCX     H               ;decrement D,H pointer
        INX     B
```

```
        LDA     TEMP2
        DCR     A               ;is all number over
        JNZ     LOOP4           ;no :loopA to square next number
        CALL    ADD32           ;yes:call ADD3 to get the addition of numbers
        CALL    SQRT            ;call SQRT
        LXI     H,0004H
        SHLD    MEM90
        CALL    DIV
        LHLD    MDIV16          ;call dvision to get rms value
        SHLD    MEM4
        XCHG
        LHLD    MRMS1
        MOV     M,E
        INX     H
        MOV     M,D
        INX     H
        SHLD    MRMS1           ;save the rms value in MRMS1
        CALL    BINTA           ;convert the value to ASCII
        POP     PSW
        POP     D
        POP     B
        POP     H               ;restore registers
        RET

        ;This routine multipies the contents of register pair DE by the
        ;contents of register pair BC. The 32 bit result will be obtained

T16     PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        MVI     A,10H           ;load the acc. with no. of bits in multiplier
        LXI     H,MEM86
        MOV     E,M
        INX     H
        MOV     D,M             ;get the multiplier in DE
        LXI     H,MEM88
        MOV     C,M
        INX     H
        MOV     B,M             ;multiplier in BC
        LXI     H,0000H         ;set register pair H and the last entry on stack
        PUSH    H
BIT     XCHG                    ;get the multiplier into H&L
        DAD     H               ;rotate the MSB into the carry
        XCHG                    ;put the multiplier back into DE


        JNC     NOADD           ;If carry = 0, don't add. The multiplicand
                                ;to the partial result and the stack
        DAD     B               ;if 1, add BC to HL, result in HL
```

```
                                ;if 1, add BC to HL, result in HL
        JNC     NOADD           ;should a 1 be added to the MSB's of the
                                ;result stored on the stack ?
        XTHL                    ;Yes, exchange HL and the stack entry.
        INX     H               ;increment the 16-bit MSW by 1
        XTHL                    ;then save it back on the stack
NOADD   DCR     A               ;decrement the bit count
        JNZ     NOTEND          ;the count is non zero, so test another bit
                                ;of multiplier.
        POP     D               ;pop the 16-bit result.
        PUSH    H
        POP     B
        LHLD    MMULT16         ;initialise the memory pointer

        MOV     M,C
        INX     H
        MOV     M,B
        INX     H
        MOV     M,E
        INX     H
        MOV     M,D             ;store the result in memory
        INX     H
        SHLD    MMULT16
        POP     PSW
        POP     D
        POP     B
        POP     H               ;restore registers
        RET
NOTEND  DAD     H               ;rotate LSW of the result left
        XTHL                    ;get the MSW into HL
        PUSH    PSW             ;save the count and carry on the stack
        DAD     H               ;rotate the MSW once to the left
        POP     PSW             ;pop the count and carry off of the stack
        JNC     NOMSB           ;was there a carry from the LSW ? no, then
                                ; do not add 1 to MSW
        INX     H               ;increment the MSW by 1
NOMSB   XTHL                    ;put the MSW on the stack
        JMP     NXTBIT          ;and test another bit in the multiplier

;this routine adds 2Ø bit no. which is stored in sequenced
;memory location and result ins 32 bit

ADD32   PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW             ;save registers
        LHL
```

```
       D       MEM6                    ;initialise the mem. pointer
       MVI     A,10H                   ;initialise the counter
       LXI     B,0                     ;initialise  register pair B & D with zero a
       LXI     D,0
CONT3  STA     TEMP1                   ;save count
       MOV     A,M
       ADD     E
       MOV     E,A
       INX     H
       MOV     A,M
       ADC     D
       MOV     D,A
       INX     H
       MOV     A,M
       ADC     C
       MOV     C,A
       INX     H
       MOV     A,M
       ADC     B
       MOV     B,A
       INX     H                       ;add partial rsult with 32 bit no.
       LDA     TEMP1                   ;load and decrement the count
       DCR     A
       JNZ     CONT3                   ;is count zero ,no:loop to add remaining no.
       LHLD    MADD32                  ;yes:initialise pointer from where 32 bit no
       MOV     M,E
       INX     H
       MOV     M,D
       INX     H
       MOV     M,C
       INX     H
       MOV     M,B                     ;store no. in memory
       INX     H
       SHLD    MADD32                  ;save the pointer
       POP     PSW
       POP     B
       POP     D
       POP     H                       ;retrieve reg.s
       RET                             ;this routine calculates the average of N 10

       ;this routine calculates the average of N 10-bit numbers.

AVRAG  PUSH    PSW
       PUSH    H
       PUSH    B
       PUSH    D                       ;s
```

```
                                        ave registers
        EXTRN   COUNT1
        LDA     MEM94A              ;load the count in acc.
        LHLD    MEM95               ;initialise memory pointer
        MOV     C,L
        MOV     B,H                 ;save the pointer in BC reg. pair
        LXI     H,Ø                 ;initialise partial result with zero
        PUSH    H                   ;save result
CONTØ   STA     MEM94A
        LDAX    B
        MOV     E,A
        INX     B
        LDAX    B
        MOV     D,A                 ;load 2 byte in DE reg pair
        POP     H
        DAD     D                   ;add the 16 bit no. to partial result
        PUSH    H
        INX     B
        LDA     MEM94A
        DCR     A
        JNZ     CONTØ               ;check if all the no.s addtion is over.no:loo
        MVI     H,Ø                 ;yes
        LDA     COUNT1
        MOV     L,A
        SHLD    MEM9Ø
        POP     H
        SHLD    MEM92
        CALL    DIV                 ;call 32 bit by 16 division routine
        LHLD    MDIV16              ;get the avg. value in memory
        SHLD    MEM4                ;save in mem4
        XCHG
        LHLD    MAVRAGØ             ;initialise the pointer from where avg. value
        MOV     M,E
        INX     H
        MOV     M,D
        INX     H                   ;store the value
        SHLD    MAVRAGØ             ;save the pointer
        CALL    BINTA               ;call BIN to ASCII routine which converts no.
        POP     D
        POP     B
        POP     H
        POP     PSW                 ;retrieve reg.
        RET                         ;return

        ;this routine divides the 16-bit quantity in register pair DE by the
        ;16-bit quantity in register pair BC

DIV
```

```
        PUSH    PSW
        PUSH    H
        PUSH    B
        PUSH    D               ;save registers
        LXI     H,TEMP4
        LXI     D,MEM9Ø         ;initialise DE with divisor mem loc.
        LDAX    D
        MOV     M,A
        INX     H
        INX     D
        LDAX    D
        MOV     M,A
        INX     H               ;save the divisor in mem
        MVI     M,11H           ;save the count 17 decimal in memory
        LXI     B,ØØØØH
        LXI     H,MEM92
        MOV     E,M
        INX     H
        MOV     D,M
NXTBIT1 LXI     H,TEMP4+2       ;load register pair H with the memmory
                                ;addresses where the bit count is stored
        MOV     A,E             ;get the LS Byte of the dividend into A
        RAL                     ;rotate the MSB into the carry
        MOV     E,A             ;save the LS byte of the dividend back in E
        MOV     A,D             ;get the MS byte of the dividend into A
        RAL                     ;rotate the MSB into the carry
        MOV     D,A             ;save the dividend's MS byte in D
        DCR     M               ;decrement the bit count
        JNZ     CONT1           ;if count not equal to zero, jump to CONT1
        MOV     L,E
        MOV     H,D
        SHLD    MDIV16          ;save the result in memory
        POP     D
        POP     B
        POP     H
        POP     PSW             ;retrieve  reg.
        RET
CONT1   MOV     A,C             ;rotate the MSB of the dividend into the parti
        RAL                     ;dividend stored in registers B and C
        MOV     C,A
        MOV     A,B
        RAL
        MOV     B,A
        DCX     H               ;decrement the memory address so that HL
        DCX     H               ;points to the divisor in memory
        MOV     A,C             ;get the LS Byte of the partial dividend
        SU
```

```
        B       M               ;subtract the LS byte of the divisor
        MOV     C,A             ;save the result back in C
        INX     H               ;increment the address
        MOV     A,B             ;get the MS Byte of the partial dividend
        SBB     M               ;subtract with borrow the divisor in memory
        MOV     B,A             ;save the result in B
        JNC     NOADD1          ;if the carry is zero, do not add the divisor
                                ;to the result of the previous subtrction
        DCX     H               ;the divisor is larger than the partial
        MOV     A,C             ;dividend,so the divisor must be added to the
        ADD     M               ;result of subtraction so that the previous
        MOV     C,A             ;value of the partial dividend is reestablishe
        INX     H
        MOV     A,B
        ADC     M
        MOV     B,A
NOADD1  CMC                     ;complement the carry
        JMP     NXTBIT1         ;then test another bit in divisor

        ;this routine calculates the  square root of 32 bit no.

SQRT    PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        LXI     H,8ØØØH         ;set MSB of shift counter
        LXI     B,ØØØØH         ;clear the bin value
SQRT1   MOV     A,C             ;get a binary value set a bit in C
        ORA     L
        MOV     C,A
        MOV     A,B
        ORA     H
        MOV     B,A             ;get binary value set a bit in B
        CALL    SQRBC           ;square binary value in BC reg. pair
        LDA     MEM1Ø7
        MOV     D,A
        LDA     MEM1Ø3
        CMP     D
        JC      RST
        JNZ     SHFT
        LDA     MEM1Ø6
        MOV D,A
        LDA     MEM1Ø2
        CMP     D
        JC      RST
        JNZ     SHFT
```

```
        LDA     MEM105
        MOV     D,A
        LDA     MEM101
        CMP     D
        JC      RST
        JNZ     SHFT
        LDA     MEM104
        MOV     A,D
        LDA     MEM100
        CMP     D
        JNC     SHFT
RST     MOV     A,C
        XRA     L
        MOV     C,A
        MOV     A,B
        XRA     H
        MOV     B,A
SHFT    MOV     A,H
        RAR
        MOV     H,A
        MOV     A,L
        RAR
        MOV     L,A
        JNC     SQRT1
        CALL    SQRBC
        LDA     MEM104
        MOV     D,A
        LDA     MEM100
        SUB     D
        CMP     C
        JC      DONE
        JZ      DONE
        INR     C
DONE    LHLD    MEMA
        MOV     M,C
        INX     H
        MOV     M,B
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET

        ;this routine squares the contents of BC register pair
        ;and result is stored in memory
        ;input-rp
```

(B)

```
        ;output-MEM1Ø4,MEM1Ø5,MEM1Ø6,MEM1Ø7(ms-BYTE)
SQRBC   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        MOV     D,B
        MOV     E,C
        MVI     A,1ØH
        LXI     H,ØØØØH
        PUSH    H
NXTBIT4 XCHG                    ;get the multiplier into H&L
        DAD     H               ;rotate the MSB into the carry
        XCHG                    ;put the multiplier back into DE
        JNC     NOADD4          ;If carry = Ø, don't add. The multiplicand
                                ;to the partial result and the stack
        DAD     B               ;if 1, add BC to HL, result in HL
        JNC     NOADD4          ;should a 1 be added to the MSB's of the
                                ;result stored on the stack ?
        XTHL                    ;Yes, exchange HL and the stack entry.
        INX     H               ;increment the 16-bit MSW by 1
        XTHL                    ;then save it back on the stack
NOADD4  DCR     A               ;decrement the bit count
        JNZ     NOTEND4         ;the count is non zero, so test another bit
                                ;of multiplier.
        POP     D               ;pop the 16-bit result.
        JNZ     NOTEND4         ;the count is non zero, so test another bit
                                ;of multiplier.
        POP     D               ;pop the 16-bit result.
        MOV     A,D
        STA     MEM1Ø7
        MOV     A,E
        STA     MEM1Ø6
        MOV     A,H
        STA     MEM1Ø5
        MOV     A,L
        STA     MEM1Ø4          ;save the result in memory
        POP     PSW
        POP     D
        POP     B
        POP     H               ;restore registers
        RET
NOTEND4 DAD     H               ;rotate LSW of the result left
        XTHL                    ;get the MSW into HL
        PUSH    PSW             ;save the count and carry on the stack
        DA
```

```
        D       H               ;rotate the MSW once to the left
        POP     PSW             ;pop the count and carry off of the stack
        JNC     NOMSB4          ;was there a carry from the LSW ? no, then
                                ; do not add 1 to MSW
        INX     H               ;increment the MSW by 1
NOMSB4  XTHL                            ;put the MSW on the stack
        JMP     NXTBIT4         ;and test another bit in the multiplier

        ;this routine converts the binary number in HL pair to
        ;its ASCII equivalent

BINTA   PUSH    H
        PUSH    B
        PUSH    D               ;save registers
        PUSH    PSW
        LHLD    MBINTA          ;load HL with the address where the
                                ;result will be stored
        XCHG                    ;save address in DE
        LXI     H,MPOINT
        MOV     M,E
        INX     H
        MOV     M,D             ;save the address in memory
        MVI     B,03H           ;load the count in B
        LHLD    MEM4            ;load binary number in HL


        LXI     D,000AH         ;place powers of te       LXI     D,000AH
        PUSH    D
        LXI     D,0064H
        PUSH    D
        LXI     D,03E8H
        PUSH    D
LOOP1   POP     D               ;get power of ten of digit to be computed
        CALL    DIGIT           ;subroutine returns digit in C
        PUSH    H               ;save binary difference
        LHLD    MPOINT          ;get pointer to digit storage area
        MOV     M,C             ;store digit
        INX     H               ;increment pointer
        SHLD    MPOINT          ;store pointer
        POP     H               ;get binary difference
        DCR     B
        JNZ     LOOP1           ;more than one digit must still be determined
        MOV     A,L
        ADI     30H
        MOV     C,A
        LHLD    MPOINT
```

```
                                ;get pointer to digit storage area
        MOV     M,C             ;store digit
        INX     H
        INX     H
        INX     H
        SHLD    MBINTA          ;save the pointer
        POP     PSW
        POP     D
        POP     B
        POP     H               ;retrieve registers
        RET
DIGIT   MVI     C,ØFFH          ;initialise C to -1
AGAIN   INR     C
        MOV     A,L             ;subtract lower order power
        SUB     E               ;of ten from binary number
        MOV     L,A             ;subtract higher order power of ten from
                                ;binary number
        MOV     A,H
        SBB     D
        MOV     H,A
        JNC     AGAIN           ;is difference positive, go back to subtract
                                ; again
        DAD     D               ;is difference negative, restore
        MOV     A,C
        ADI     3ØH
        MOV     C,A             ;convert the digit into ASCII
        RET

        ;this routine finds the maximum number ( 1Ø-bit) from a string of
        ;N numbers

MAX     PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        MVI     A,32H
        STA     MEM94B
        LDA     MEM94B          ;initialise count 5Ø decimal to MEM 94B and
        MOV     B,A             ;store it in register B
        LXI     D,Ø             ;initialise maximum number in DE register pair
        LHLD    MEM95           ;initialise memory pointer
CONT2   MOV     A,M




        SUB     E
```

```
        INX     H
        MOV     A,M             ;compare number with maximum number
        SBB     D
        JC      LOOP2           ;is number > maximum number, NO jump to loop2
        MOV     D,M             ;yes, store number as maximum number
        DCX     H
        MOV     E,M
        INX     H
LOOP2   INX     H               ;compare with next number
        DCR     B
        JNZ     CONT2           ;is comparison over,NO jumpto cont2
        XCHG
        SHLD    MEM4            ;YES, store the number in memory
        XCHG
        LHLD    MMAX1
        MOV     M,E
        INX     H
        MOV     M,D             ;store the number in MAX1
        INX     H
        SHLD    MMAX1
        LHLD    MAXMAX          ;load HL wit the address of the higher limit
                                ;of the number
        MOV     A,M
        SUB     E
        INX     H
        MOV     A,M
        SBB     D
        JNC     LOOPØ2
        LDA     BYTEMAX
        MOV     B,A
        LDA     BYTEST
        ORA     B
        STA     BYTEMAX
LOOPØ2  LDA     BYTEST
        RLC
        STA     BYTEST
        INX     H
        SHLD    MAXMAX
        CALL    BINTA
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
```

```
IN      PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        MVI     A,32H
        STA     MEM94C
        LDA     MEM94C          ;initialise count 50 decimal to MEM 94C and
        MOV     B,A             ;store it in register B
        LXI     D,0FFFFH        ;initialise minimum number in DE register pair
        LHLD    MEM95           ;initialise memory pointer
ONT5    MOV     A,M
        SUB     E
        INX     H
        MOV     A,M             ;compare number with minimum number
        SBB     D
        JNC     LOOP5           ;is number < minimum number. NO jump to loop5
        MOV     D,M             ;yes, store number as minimum number
        DCX     H
        MOV     E,M
        INX     H
OOP5    INX     H               ;compare with next number
        DCR     B
        JNZ     CONT5           ;is comparison over.NO jumpto cont5
        XCHG
        SHLD    MEM4            ;YES, store the number in memory
        XCHG
        LHLD    MMIN1
        MOV     M,E
        INX     H
        MOV     M,D             ;store the number in MIN1
        INX     H
        SHLD    MMIN1
        LHLD    MINMIN          ;load HL wit the address of the lower limit
                                ;of the number
        MOV     A,M
        SUB     E
        INX     H
        MOV     A,M
        SBB     D               ;compare if number is less than lower limit
        JC      LOOP05          ;NO, jump to loop05
        LDA     BYTEMIN
        MOV     B,A
        LDA     BYTEST0         ;YES, make that bit 1 in bytemin location
        ORA     B
        STA     BYTEMIN
OOP05   LDA     BYTEST0
        RLC
```

```
                                  ;shift the bit
        STA     BYTESTØ
        INX     H
        SHLD    MINMIN            ;store the addresss of next lower limit
        CALL    BINTA             ;convert the minimum number into ASCII
        POP     PSW
        POP     D
        POP     B
        POP     H                 ;restore registers
        RET

        ;displays the status of digital input in the form of ON or OFF
        ;input port-A of 8255-1

STATUS  PUSH    H
        PUSH    B

        PUSH    D
        PUSH    PSW               ;save registers
        LXI     H,MSTATUS         ;initialise the memory pointer and save its
        SHLD    TEMPS             ;value in temps
        MVI     A,9BH             ;initialisation of 8255-1
        OUT     53H               ;all input ports
        MVI     C,Ø8H             ;initialise count
        MVI     B,Ø1H             ;save shift bit count in reg. B
        IN      5ØH               ;read the status
        STA     TEMPSØ            ;store its value
CONT6   LDA     TEMPSØ            ;load status in accumulator
        ANA     B                 ;check if bit is Ø
        JZ      OFF               ;YES, jump to store code of OFF
        LHLD    TEMPS
        MVI     A,2ØH
        MOV     M,A
        INX     H
        MVI     A,4FH
        MOV     M,A
        INX     H
        MVI     A,4EH             ;store the ASCII code of " ON " in memory
        MOV     M,
```

```
        INX     H
        MVI     A,20H
        MOV     M,A
        INX     H
        SHLD    TEMPS               ;save the pointer
        JMP     LAST
OFF     LHLD    TEMPS
        MVI     A,20H
        MOV     M,A
        INX     H
        MVI     A,4FH
        MOV     M,A
        INX     H
        MVI     A,46H               ;store the ASCII code of " OFF " in memory
        MOV     M,A
        INX     H
        MOV     M,A
        INX     H
        SHLD    TEMPS               ;save the pointer
LAST    MOV     A,B
        RLC                         ;rotate left the shift bit
        MOV     B,A
        DCR     C                   ;is all 8 bits checked ?
        JNZ     CONT6               ;NO , jump to cont6
        MVI     A,0A0H
        MOV     B,A                 ;YES, save the no. of bytes to be transmitted
        LXI     H,0B3C0H            ;in B and initialise the HL with the starting
                                    ;address of the block
WAIT6   IN      0F1H
        ANI     01H
        JZ      WAIT6               ;output a byte
        MOV     A,M
        OUT     0F0H
        INX     H                   ;increment the pointer
        DCR     B
        JNZ     WAIT6               ;NO, jump to wait6
        POP     PSW
        POP     D
        POP     B
        POP     H                   ;YES, retrieve registers





        RET
```

;main display sub. to display all values(rms,avrag-etc) of 16 analog
;variable in decimal form

```
DISPLAY PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        LXI     B,03C0H         ;no. of bytes to be transmitted in BC
                                ;register pair
        LXI     H,0B000H        ;starting address of block in HL reg pair
WAIT7   IN      0F1H
        ANI     01H
        JZ      WAIT7
        MOV     A,M
        OUT     0F0H            ;transmit one byte
        INX     H
        DCX     B
        MOV     A,C
        ORA     B               ;is all bytes over ?
        JNZ     WAIT7           ;NO, jump to wait7 to transmit rest of the bytes
        POP     PSW
        POP     D
        POP     B
        POP     H               ;restore registers





        RET
```

```
        ;main display sub. to display all values(rms,avrag-etc) of 16 analog
        ;variable in decimal form

DISPLAY PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW                 ;save registers
        LXI     B,Ø3CØH             ;no, of bytes to be transmitted in BC
                                    ;register pair
        LXI     H,ØBØØØH            ;starting address of block in HL reg pair
WAIT7   IN      ØF1H
        ANI     Ø1H
        JZ      WAIT7
        MOV     A,M
        OUT     ØFØH                ;transmit one byte
        INX     H
        DCX     B
        MOV     A,C
        ORA     B                   ;is all bytes over ?
        JNZ     WAIT7               ;NO, jump to wait7 to transmit rest of the bytes
        POP     PSW
        POP     D
        POP     B
        POP     H                   ;restore registers
        RET

        ;i.s.s-this gives the command to start swt.device

IRØ     PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW                 ;save registers
        MVI     E,ØØH               ;initialise bit counter in E and shift
        MVI     B,Ø1H               ;count in B
        IN      51H                 ;input from port and store in memory
        STA     MEM16
AGAINØØ LDA     MEM16
        ANA     B                   ;check interrupt from which device NO, jump
                                    ;to check next bit
        JZ      CHECKØØ
        MOV     A,E
        STA     MEM17               ;YES, save the bit no. in MEM17
        CALL    DISP1               ;call display to  display comand
CHECKØØ INR     E                   ;increment bit count
        MOV     A,B
        RLC                         ;shift count
        MOV
```

```
                B,A
        ANI     Ø1H             ;all bits checked ?
        JZ      AGAINØØ         ;NO, jump to againØØ
        POP     PSW
        POP     D
        POP     B
        POP     H               ;YES, restore registers
        RET
DISP1   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        LXI     H,ØB5ØØH        ;initialise starting address of the block
        MVI     B,1EH           ;save the no. of bytes to be transmitted in B
WAITØØ  IN      ØF1H
        ANI     Ø1H
```

```
        JZ      WAITØØ
        INX     H
        MOV     A,M
        OUT     ØFØH            ;transmit one byte
        DCR     B               ;decrement count
        JNZ     WAITØØ          ;is all bytes transmitted ? NO,transmit next
        LDA     MEM17
        ADI     3ØH
        MOV     B,A
WAITØ1  IN      ØF1H
        ANI     Ø1H
        JZ      WAITØ1
        MOV     A,B
        OUT     ØFØH            ;transmit the bit no.
WAITØ2  IN      ØF1H
        ANI     Ø1H
        JZ      WAITØ2
        MVI     A,ØDH
        OUT     ØFØH            ;transmit CR code
WAITØ3  IN      ØF1H
        ANI     Ø1H
        J
```

```
        Z       WAIT03
        MVI     A,0AH
        OUT     0F0H            ;transmit LF code
        POP     PSW
        POP     D
        POP     B
        POP     H               ;restore registers
        RET

        ;IR1, IR2,IR3 are exactly similar procedures but each one transmits
        ;a sligtly different command

        ;i.s.s-this gives the command to start plant
IR1     PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        MVI     E,00H
        MVI     B,01H
        IN      52H             ;port-c of 8255-1
        STA     MEM16
AGAIN01 LDA     MEM16
        ANA     B
        JZ      CHECK01
        MOV     A,E
        STA     MEM17
        CALL    DISP2
CHECK01 INR     E
        MOV     A,B
        RLC
        MOV     B,A
        ANI     01H
        JZ      AGAIN01
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
DISP2   PUSH    H
        PUSH    B
        PUSH    D




        PUSH    PSW             ;save registers
```

```
         LXI     H,0B600H
         MVI     B,18H
WAIT10   IN      0F1H
         ANI     01H
         JZ      WAIT10
         INX     H
         MOV     A,M
         OUT     0F0H
         DCR     B
         JNZ     WAIT10
         LDA     MEM17
         ADI     30H
         MOV     B,A
WAIT11   IN      0F1H
         ANI     01H
         JZ      WAIT11
         MOV     A,B
         OUT     0F0H
WAIT12   IN      0F1H
         ANI     01H
         JZ      WAIT12
         MVI     A,0DH
         OUT     0F0H
WAIT13   IN      0F1H
         ANI     01H
         JZ      WAIT13
         MVI     A,0AH
         OUT     0F0H
         POP     PSW
         POP     D
         POP     B
         POP     H
         RET
         ;I.S.S- this interrups when any swt. device has to be stopped
IR2      PUSH    H
         PUSH    B
         PUSH    D
         PUSH    PSW
         MVI     E,00H
         MVI     B,01H
         IN      54H             ;PORT-A OF 8255-2
         STA     MEM16
AGAIN02  LDA     MEM16
         ANA     B
         JZ      CHECK02
         MOV     A,E
         STA     MEM17
         CALL
```

```
                DISP3
CHECK02  INR    E
         MOV    A,B
         RLC
         MOV    B,A
         ANI    01H
         JZ     AGAIN02
         POP    PSW
         POP    D
         POP    B
         POP    H
         RET
DISP3    PUSH   H
         PUSH   B
```

```
         PUSH   D
         PUSH   PSW
         LXI    H,0B700H
         MVI    B,1DH
WAIT20   IN     0F1H
         ANI    01H
         JZ     WAIT20
         INX    H
         MOV    A,M
         OUT    0F0H
         DCR    B
         JNZ    WAIT20
         LDA    MEM17
         ADI    30H
         MOV    B,A
WAIT21   IN     0F1H
         ANI    01H
         JZ     WAIT21
         MOV    A,B
         OUT    0F0H
WAIT22   IN     0F1H
         ANI    01H
         JZ     WAIT22
```

```
        MVI     A,ØDH
        OUT     ØFØH
WAIT23  IN      ØF1H
        ANI     Ø1H
        JZ      WAIT23
        MVI     A,ØAH
        OUT     ØFØH
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        EXTRN   MADC,CHNUM
ADCREAD PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW             ;save registers
        LDA     CHNUM
        OUT     ØBØH            ;initialise channel
        CALL    DELAY           ;wait for 2Øms so adc conversion is over
WAIT2   IN      ØB2H
        RLC
        JC      WAIT2           ;is conversion over?NO,wait
        XRA     A               ;YES,read higher byte and store its 2 MSB'S
        IN      ØB1H            ;in register h and LSB'S in register b
        RAL
        MOV     B,A
        MVI     A,Ø
        RAL
        MOV     C,A
        XRA     A
        MOV     A,B
        RAL
        MOV     B,A
        MOV     A,C
        RAL
        MOV     H,A
        IN      ØB2H            ;read lower byte
        ANI     Ø3H             ;mask six higher order bits     ADD     B



        MOV     L,A             ;store in l registers
```

```
        SHLD    HADC                ;store digital value in memory
        POP     PSW
        POP     D
        POP     B
        POP     H                   ;restore registers
        RET
DELAY   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW                 ;save registers
        LXI     B,ØØØ2H             ;initialise the delay count in register B
LOOP8   DCX     B                   ;decrement the count
        MOV     A,C
        ORA     B                   ;check  if period is over
        JNZ     LOOP8               ;no, jump to decrement count
        POP     PSW
        POP     D
        POP     B
        POP     H                   ;restore registers
        RET

        ;display wrong code pressed

DISP4   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW                 ;save registers
        LXI     H,ØB9ØØH
        MVI     B,1DH
WAIT4Ø  IN      ØF1H
        ANI     Ø1H
        JZ      WAIT4Ø
        INX     H
        MOV     A,M
        OUT     ØFØH
        DCR     B
        JNZ     WAIT4Ø
        LDA     MEM17
        ADI     3ØH
        MOV     B,A
WAIT41  IN      ØF1H
        ANI     Ø1H
        JZ      WAIT41
        MOV     A,B
        OUT     ØFØH
WAIT42  IN      ØF1H
        AN
```

```
          I        Ø1H
          JZ       WAIT42
          MVI      A,ØDH
          OUT      ØFØH
WAIT43    IN       ØF1H
          ANI      Ø1H
          JZ       WAIT43
          MVI      A,ØAH
          OUT      ØFØH
          POP      PSW
          POP      D
          POP      B
          POP      H
          RET
          END
```

```
                ORG     4Ø96Ø
;initialisation of all memory locations
RMS1S    SET    ØD5Ø8H
AVRAG1S  SET    ØD51ØH
MAX1S    SET    ØD52ØH
MIN1S    SET    ØD53ØH
MEMINS   SET    ØBØAEH
MEMMIN   SET    ØB1EEH
MEMMAX   SET    ØB19EH
MAVRAG1  SET    ØB23EH
MPOINT   SET    94ØØH
COUNT1   SET    9492H
MEM4     SET    94Ø4H
MEM94A   SET    94Ø6H
MEM9Ø    SET    94Ø8H
MEM92    SET    94ØAH
MBINTA   SET    94ØCH
TEMP4    SET    94ØEH
MDIV16   SET    941ØH
MEM95    SET    9412H
MEM1ØØ   SET    9414H
MEM1Ø1   SET    9415H
MEM1Ø2   SET    9416H
MEM1Ø3   SET    9417H
MEM1Ø4   SET    9418H
MEM1Ø5   SET    9419H
MEM1Ø6   SET    941AH
MEM1Ø7   SET    941BH
MEM94B   SET    941CH
MEM94C   SET    941EH
MAXMAX   SET    ØA42ØH
BYTEMAX  SET    ØA422H
BYTEST   SET    ØA423H
TEMP1    SET    9424H
MEM86    SET    9426H
MEM88    SET    9428H
MEM6     SET    942AH
TEMP2    SET    942CH
MMULT16  SET    942EH
MADD32   SET    943ØH
MEMA     SET    9432H
TEMPMUL  SET    9434H
MRMS     SET    ØB32EH
MADC     SET    9436H
MINMIN   SET    ØA438H
BYTEMIN  SET    ØA43AH
BYTESTØ  SET    ØA43BH
```

```
STOREØØ  SET     9ØØØH
STOREØ4  SET     9Ø1ØH
STOREØ5  SET     STOREØ4+1ØØ
STOREØ6  SET     STOREØ5+1ØØ
STOREØ7  SET     STOREØ6+1ØØ
STOREØ8  SET     STOREØ7+1ØØ
STOREØ9  SET     STOREØ8+1ØØ
STOREØA  SET     STOREØ9+1ØØ
STOREØB  SET     STOREØA+1ØØ
STOREØC  SET     95ØØH
STOREØD  SET     STOREØC+64
STOREØE  SET     STOREØD+64
STOREØF  SET     STOREØE+64
TEMPØØ   SET     98ØØH
TEMPØ4   SET     98Ø2H
TEMPØ5   SET     98Ø4H
TEMPØ6   SET     98Ø6H
TEMPØ7   SET     98Ø8H
TEMPØ8   SET     98ØAH
TEMPØ9   SET     98ØCH
TEMPØA   SET     98ØEH
TEMPØB   SET     981ØH
TEMPØC   SET     9812H
TEMPØD   SET     9814H
TEMPØE   SET     9816H
TEMPØF   SET     9818H
TEMP1Ø   SET     981AH
TEMP2Ø   SET     981CH
TEMP3Ø   SET     981EH
MAVRAGØ  SET     982ØH
MRMS1    SET     9822H
MMAX1    SET     9824H
MMIN1    SET     9826H
CHNUM    SET     9828H
MEM16    SET     982AH
MEM17    SET     982CH
MSTATUS  SET     ØB41EH
TEMPS    SET     982EH
TEMPSØ   SET     983ØH
ADCCH    EQU     ØBØH
HIBYT    EQU     ØB1H
LOBYT    EQU     ØB2H
         ;define public and external variables
         PUBLIC  MEM16,MEM17,MSTATUS,TEMPS,TEMPSØ
         PUBLIC  MAVRAG1,STOREØ4,STOREØ5,STOREØ6,STOREØ7,STOREØ8,STOREØ9,STOREØ
         PUBLIC  STOREØB,MEMMAX,MEM94B,MEM94C,MEMMIN,BYTEMAX,MAXMAX,BYTEST,MINM
         PUB
```

```
        LIC     STOREØC,STOREØD,STOREØE,STOREØF,MEM1ØØ,MEM1Ø1,MEM1Ø2,MEM1Ø3,MEM
        PUBLIC  MEM1Ø5,MEM1Ø6,MEM1Ø7,TEMP1,MEM86,MEM88,MEM6,TEMP2,MMULT16,MEMA,
        PUBLIC  TEMP1Ø,TEMP2Ø,ADCCH,CHNUM,LOBYT,HIBYT,MADC,SLOW,FAST,AGAIN4,AGA
        PUBLIC  MPOINT,MEM4,TEMP4,MEM94A,MEM95,MEM9Ø,MEM92,MBINTA,MDIV16,COUNT1
        PUBLIC  AVRAG1S,MAX1S,MIN1S,RMS1S,MAVRAGØ,MRMS1,MMAX1,MMIN1
        EXTRN   ADCREAD,BINTA
        ;main program
        MVI     A,1DH
        SIM                     ;enable RST7.5,  RST6.5, RST5.5
        MVI     A,Ø
        STA     TEMP1Ø
        STA     TEMP2Ø          ;initialise memory contents to Ø
        MVI     A,7ØH
        OUT     ØF7H
        MVI     A,8ØH
        OUT     ØF5H            ;initialisation of counter 2 of 8253
        MVI     A,Ø7H
        OUT     ØF5H
        MVI     A,37H
        OUT     ØF7H
        MVI     A,1ØH
        OUT     ØF4H            ;initialisation of counter 1 of 8253
        MVI     A,ØØH
        OUT     ØF4H
        MVI     A,4FH
        OUT     ØF1H
        MVI     A,Ø5H
        OUT     ØF1H            ;initialisation of 8251 txd and rxd enable
        MVI     A,9BH
        OUT     53H
        OUT     57H             ;initialisation of 8255-1 and 8255-2
        MVI     A,16H
        OUT     ØF2H            ;initialisation of 8259
        MVI     A,8AH
        OUT     ØF3H            ;unmask IRØ,IR1,IR2
        MVI     A,ØF8H
        OUT     ØF3H
AGAIN1  MACRO
        LXI     H,STOREØØ
        SHLD    TEMPØØ
        ENDM
AGAIN2  MACRO
        LXI     H,STOREØ4
        SHLD    TEMPØ4
        LXI     H,STOREØ5
        SHLD    TEMPØ5
        LXI     H,STOREØ6
        SH
```

```
         LD       TEMP06
         LXI      H,STORE07          ;initialisation of memory location
         SHLD     TEMP07             ;for uncontrolled slow variables
         LXI      H,STORE08
         SHLD     TEMP08
         LXI      H,STORE09
         SHLD     TEMP09
         LXI      H,STORE0A
         SHLD     TEMP0A
         LXI      H,STORE0B
         SHLD     TEMP0B
         ENDM
AGAIN3   MACRO
         LXI      H,STORE0C
         SHLD     TEMP0C
         LXI      H,STORE0D          ;initialisation of memory location for fast
         SHLD     TEMP0D             ;variables
         LXI      H,STORE0E
         SHLD     TEMP0E
         LXI      H,STORE0F
         SHLD     TEMP0F
         ENDM
STORE    MACRO
         MOV      M,E
         INX      H
         MOV      M,D
         INX      H
         ENDM
         AGAIN1
         AGAIN2
         AGAIN3
         LXI      H,MEMINS           ;initialisation of memory for slow controlled
         SHLD     MBINTA             ;variables
         CALL     SLOW               ;call slow to input slow analog variables
         JMP      GO2                ;jump to go2
AGAIN4   PUSH     H
         PUSH     B
         PUSH     D
         PUSH     PSW
         AGAIN3
         POP      PSW
         POP      D
         POP      B
         POP      H
         RET
AGAIN5   PUSH     H
         PUSH     B
         PUSH     D
         PUSH
```

```
                PSW
        AGAIN2
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
SLOW    PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        LXI     H,MEMINS        ;initialise pointer from memory instantaneous
        SHLD    MBINTA          ;and store in memory MBINTA
        MVI     A,Ø
        STA     CHNUM           ;initialise channel no.
CONT6   CALL    ADCREAD         ;read the ADC
        LHLD    MADC
        XCHG
        LHLD    TEMPØØ
        STORE                   ;save the digital value in memory
        SHLD    TEMPØØ
        XCHG
        SHLD    MEM4
        CALL    BINTA           ;convert the binary no, into ASCII
        LDA     CHNUM
        INR     A               ;increment the channel no.
        STA     CHNUM
        CPI     Ø4H             ;is channel no. = Ø4 ?
        JZ      LOOP6           ;YES,call SLOW next
        JMP     CONT6           ;NO,jum p to input channel value
LOOP6   AGAIN1
        CALL    SLOW1           ;call slow to input variables(Ø5- 12)
        POP     PSW
        POP     D
        POP     B
        POP     H               ;restore registers
        RET
GO2     CALL    FAST            ;input fast variables
HAULT   EI                      ;enable interrupts
        HLT                     ;halt
        JMP     HAULT           ;jump to wait for next interrupts
STORE1  MACRO
        INR     B
        MOV     A,B
        STA     CHNUM
        CALL
```

```
            ADCREAD
        LHLD    MADC
        XCHG
        ENDM
SLOW1   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        MVI     A,Ø4H           ;initialise channel no to 4
        STA     CHNUM
        MOV     B,A             ;save the value in register B
        CALL    ADCREAD         ;read the ADC
        LHLD    MADC            ;store the value
        XCHG
        LHLD    TEMPØ4
        STORE
        SHLD    TEMPØ4          ;increment the channel no. to ØB and read
        STORE1                  ;adn store
        LHLD    TEMPØ5
        STORE
        SHLD    TEMPØ5
        STORE1
        LHLD    TEMPØ6
        STORE
        SHLD    TEMPØ6
        STORE1
        LHLD    TEMPØ7
        STORE
        SHLD    TEMPØ7
        STORE1
        LHLD    TEMPØ8
        STORE
        SHLD    TEMPØ8
        STORE1
        LHLD    TEMPØ9
        STORE
        SHLD    TEMPØ9
        STORE1
        LHLD    TEMPØA
        STORE
        SHLD    TEMPØA
        STORE1
        LHLD    TEMPØB
        STORE
        SHLD
```

```
                TEMPØB
        POP     PSW
        POP     D
        POP     B
        POP     H                       ;resotre registers
        RET
FAST    PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        MVI     A,ØCH
        STA     CHNUM                   ;initialise chanel no. to ØCH
        MOV     B,A                     ;save the value in register B
        CALL    ADCREAD                 ;read the ADC
        LHLD    MADC                    ;store the value
        XCHG
        LHLD    TEMPØC
        STORE
        SHLD    TEMPØC
        STORE1
        LHLD    TEMPØD
        STORE
        SHLD    TEMPØD
        STORE1
        LHLD    TEMPØE
        STORE
        SHLD    TEMPØE                  ;increment the channel no. to ØFH and
        STORE1                          ;read and store their value
        LHLD    TEMPØF
        STORE
        SHLD    TEMPØF
        POP     PSW
        POP     D
        POP     B
        POP     PSW                     ;restore registers
        RET
        END
```

```
        ORG     32848
        PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        EXTRN   MRMS1,RMS1S,STATUS,DISPLAY
        EXTRN   MINMIN,BYTEMIN,BYTESTØ,MMIN1,MIN1S,MAX1S,MMAX1,MAVRAGØ,AVRAG1S
        EXTRN   TEMPMUL,MEM6,MMULT16,MAVRAG1,STOREØ4,STOREØ5,STOREØ6,STOREØ7,S1
        EXTRN   STOREØA,STOREØB,MEMMAX,MAX,MEMMIN,MIN,MAXMAX,BYTEMAX,BYTEST
        EXTRN   STOREØC,STOREØD,STOREØE,STOREØF,MEM94A,MEM95,MEM1ØØ,MEMA,RMS,M/
        EXTRN   AVRAG,TEMP1Ø,TEMP2Ø,CHNUM,SLOW,FAST,AGAIN4,AGAIN5,COUNT1,MBINT/
        MVI     A,8ØH
        OUT      ØF5H
        MVI     A,Ø7H
        OUT     ØF5H                ;relode the counter
        LDA     TEMP1Ø
        INR     A
        STA     TEMP1Ø              ;increment the count stored in TEMP1Ø
        CPI     1ØH                 ;check if 2ØmS period over
        JNZ     GO1                 ;NO,jump to GO1
        LDA     TEMP2Ø              ;YES,increment the counter which counts the 1s
        INR     A
        STA     TEMP2Ø
        CPI     32H                 ;is 1sec over?
        JZ      GO2                 yes,jump to calculate MAX,MIN,AVRAG of slow
                                    ;variable
        CALL    LESS                ;call LESS
        CALL    AGAIN4              ;initialise memory for fast variables
        JMP     ENDD                ;jump to last
LESS    PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW                 ;save registers
        XRA     A
        STA     TEMP1Ø              ;clear memory
        CALL    SLOW                ;call SLOW to input slow variables
        LXI     H,STOREØC
        SHLD    MEM95
        LXI     H,MRMS
        SHLD    MBINTA
        LXI     H,RMS1S
        SHLD    MRMS1
        CALL    RMS                 ;calculate RMS value of four fast variables
        LXI
```

```
                H,STOREØD           ;and conert them into ASCII and store
        SHLD    MEM95
        CALL    RMS
        LXI     H,STOREØE
        SHLD    MEM95
        CALL    RMS
        LXI     H,STOREØF
        SHLD    MEM95
        CALL    RMS
        POP     PSW
        POP     D
        POP     B
        POP     H                   ;restore registers
        RET
GO1     CALL    FAST                ;call FAST to input fast variables
        JMP     ENDD                ;jump to end
GO2     MVI     A,32H               ;set count
        STA     COUNT1
        STA     MEM94A
        LXI     H,MAVRAG1
        SHLD    MBINTA
        LXI     H,STOREØ4
        SHLD    MEM95
        LXI     H,AVRAG1S
        SHLD    MAVRAGØ
        CALL    AVRAG
        MVI     A,32H
        STA     MEM94A
        LXI     H,STOREØ5
        SHLD    MEM95
        CALL    AVRAG               ;calculate AVERAGE value of eight slow variable
        MVI     A,32H               ;and conert them into ASCII and store
        STA     MEM94A
        LXI     H,STOREØ6
        SHLD    MEM95
        CALL    AVRAG
        MVI     A,32H
        STA     MEM94A
        LXI     H,STOREØ7
        SHLD    MEM95
        CALL    AVRAG
        MVI     A,32H
        STA     MEM94A
        LXI     H,STOREØ8
        SHLD    MEM
```

```
CALL    AVRAG
MVI     A,32H
STA     MEM94A
LXI     H,STOREØ9
SHLD    MEM95
CALL    AVRAG
MVI     A,32H
STA     MEM94A
LXI     H,STOREØA
SHLD    MEM95
CALL    AVRAG
MVI     A,32H
STA     MEM94A
LXI     H,STOREØB
SHLD    MEM95
CALL    AVRAG
LXI     H,9FØØH
SHLD    MAXMAX
MVI     A,ØØH
STA     BYTEMAX
MVI     A,Ø1H
STA     BYTEST
LXI     H,MEMMAX
SHLD    MBINTA
LXI     H,STOREØ4
SHLD    MEM95
LXI     H,MAX1S
SHLD    MMAX1
CALL    MAX
LXI     H,STOREØ5
SHLD    MEM95
CALL    MAX
LXI     H,STOREØ6
SHLD    MEM95
CALL    MAX
LXI     H,STOREØ7
SHLD    MEM95          ;calculate MAX value of eightslow variables
CALL    MAX            ;and conert them into ASCII and store
LXI     H,STOREØ8
SHLD    MEM95
CALL    MAX
LXI     H,STOREØ9
SHLD    MEM95
CALL    MAX
LXI     H,STO
```

```
                RE0A
        SHLD    MEM95
        CALL    MAX
        LXI     H,STORE0B
        SHLD    MEM95
        CALL    MAX
        LXI     H,9F10H
        SHLD    MINMIN
        MVI     A,00H
        STA     BYTEMIN
        MVI     A,01H
        STA     BYTEST0
        LXI     H,MEMMIN
        SHLD    MBINTA
        LXI     H,STORE04
        SHLD    MEM95
        LXI     H,MIN1S
        SHLD    MMIN1
        CALL    MIN
        LXI     H,STORE05
        SHLD    MEM95
        CALL    MIN
        LXI     H,STORE06
        SHLD    MEM95
        CALL    MIN
        LXI     H,STORE07
        SHLD    MEM95
        CALL    MIN
        LXI     H,STORE08
        SHLD    MEM95
        CALL    MIN             ;calculate MIN value of eightslow variables
        LXI     H,STORE09       ;and conert them into ASCII and store
        SHLD    MEM95
        CALL    MIN
        LXI     H,STORE0A
        SHLD    MEM95
        CALL    MIN
        LXI     H,STORE0B
        SHLD    MEM95
        CALL    MIN
        CALL    LESS            ;call LESS to calculate RMS value of four fast
                                ;vairables and conert them into ASCII and stor
        CALL    DISPLAY         ;call DISPLAY to display processed data
        CALL    STATUS          ;call status to display status of switching de
        JMP     GO3
ENDD    POP     PSW
        POP     D
        POP     B


        POP     H               ;restore registers
        RET
GO3     CALL    AGAIN5          ;initialise memory for slow variables
        JMP     ENDD
        END
```

```
        ORG     35328
        EXTRN   IRØ,IR1,IR2,DISPLAY,STATUS
;initilaise memory block for interrupts of 8259
        JMP     J1
        JMP     J2
        JMP     J3
        JMP     J4
        JMP     J4
        JMP     J4
        JMP     J4
J4      PUSH    PSW
        MVI     A,2ØH
        OUT     ØF2H            ;eoi command
        POP     PSW
        EI
        RET
J1      CALL    IRØ             ;call IRØ to display start switching device
        CALL    DISPLAY         ;call display to display analog data
        CALL    STATUS          ;call status to display status of switching
                                ;devices
        PUSH    PSW
        MVI     A,2ØH
        OUT     ØF2H            ;eoi command
        POP     PSW
        EI
        RET
J2      CALL    IR1             ;call IR1 to display start plant
        CALL    DISPLAY         ;call display to display analog data
        CALL    STATUS          ;call status to display status of switching
                                ;devices
        PUSH    PSW
        MVI     A,21H
        OUT     ØF2H            ;eoi command
        POP     PSW
        EI
        RET
J3      CALL    IR2             ;call IR2 to display stop switching device
        CALL    DISPLAY         ;call display to display analog data
        CALL    STATUS          ;call status to display status of switching
                                ;devices
        PUSH    PSW
        MVI     A,22H
        OUT     ØF2H            ;eoi command
        POP     PSW
        EI
        RET
```

```
        ORG     6000H
EXTERN  READ00, READ01, MEM17, DISP3, DISP2, DISP1, DISP4, SUB38, SUB39
EXTERN  WRONG
;this   program takes command from operator and takes action accordingly
        PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    3C0CH   ;read keyboard
        CPI     01H
        JNZ     STEP03
        CALL    SUB01
        JMP     DO180   ;check if type is 01, if yes : call SUB01 and jump to LA
STEP03  CPI     02H
        JNZ     STEP04
        CALL    SUB02
        JMP     DO180   ;check if type is 02, if yes : call SUB01 and jump to LA
STEP04  CPI     03H
        JNZ     STEP05
        CALL    SUB03
        JMP     DO180   ;check if type is 03, if yes : call SUB01 and jump to LA
STEP05  CPI     04H
        JNZ     STEP06
        CALL    SUB04
        JMP     DO180   ;check if type is 04, if yes : call SUB01 and jump to LA
STEP06  CPI     05H
        JNZ     STEP07
        CALL    SUB05
        JMP     DO180   ;check if type is 05, if yes : call SUB01 and jump to LA
STEP07  CPI     06H
        JNZ     STEP08
        CALL    SUB06
        JMP     DO180   ;check if type is 06, if yes : call SUB01 and jump to LA
STEP08  CPI     07H
        JNZ     STEP09
        CALL    SUB07
        JMP     DO180   ;check if type is 07, if yes : call SUB01 and jump to LA
STEP09  CPI     08H
        JNZ     STEP010
        CALL    SUB08
        JMP     DO180   ;check if type is 08, if yes : call SUB01 and jump to LA
STEP010 CPI     09H
        JNZ     STEP011
        CALL    SUB09
        JMP     DO180   ;check if type is 09, if yes : call SUB01 and jump to LA
STEP011 CPI     0AH
        JNZ
```

```
                STEPØ12
        CALL    SUBØA
        JMP     DO18Ø    ;check if type is ØA, if yes : call SUBØ1 and jump to LA
TEPØ12  CPI     ØBH
        JNZ     STEPØ13 ;check if  ØB  is pressed, if yes call SUBØB and  jump t
        CALL    SUBØB
        JMP     DO18Ø
STEPØ13 CALL    WRONG    ;if no call display "Wrong code pressed"
DO18Ø   POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        ;this routine again  reads the  plant no. and displays to start the  pl
SUBØ1   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    3CØCH    ;read plant no.
        STA     MEM17
        CALL    DISP2    ;store  the  accumulator in MEM17 and call display
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        ;this routine again  reads the  plant no. and displays to stop the  pla
SUBØ2   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    3CØCH    ;read the plant number
        STA     MEM17
        CALL    DISP4    ;store the  accumulator in MEM17 and call display
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        ;this routine reads the switch no. and displays  to starts that device
SUBØ3   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    3CØCH
        STA     MEM17    ;read switch device no. and  store in MEM17
        CAL
```

```
        L       DISP1    ;call display
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        ;this routine resets  the  counter Ø to FFFFH
SUBØ4   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    3CØCH
        STA     MEM17
        CALL    DISP3
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        ;this routine resets  the  counter Ø to FFFFH
SUBØ8   CALL    SUB38
        RET
        ;this routine resets  the  counter 1 to FFFFH
SUBØ9   CALL    SUB39
        RET
        ;this routine reads  the  counter Ø to FFFFH
SUBØA   CALL    READØØ
        RET
        ;this routine reads  the  counter 1 to FFFFH
SUBØB   CALL    READØ1
        RET




        END



        END
```

APPENDIX-B

```
c         SUBROUTINE ABC
         COMMON/COUNTE/NK,NK5,NOVR,NL,NOTV,NPHASE,NUSIS,NOTIV,NL4
         COMMON/SYS1/N1,N2,IOU1,IBUS,IBAR,NVX2,IHX1,IHY1,IHX2,IHY2,N,IOBI
        1,IOG1,B1,B2,B3,G1
         DIMENSION B(10),V1(10),XA1(10),IX1(10),IY1(10),IW(10),IVX1(10)
        1,IVY1(10),INX1(10),INY1(10),IBX1(10),IBY1(10),TYPE(10),IG1(10)
        2,IRX1(10),IRY1(10),icx(20),icy(20),itx(20),ity(20)
         DIMENSION BA(20),BB(20),IOV(20),IMVA(20),IX2(20),IY2(20),IX3(20)
        1,IY3(20),IX4(20),IY4(20),IX5(20),IY5(20)
         CHARACTER * 8 B,BA,BB,B1,B2,B3
         CHARACTER * 4 V1
         CHARACTER *56 N
         CHARACTER *5 XA1,G1
         CHARACTER *3 CCB(20),CB
         character *2 trna(20)
         nvx1=14
         NVX2=15
         N1=5
         N2=7
         IOU1=1
         IOBI=1
         IOG1=0
         IBUS=10
         IBAR=20
         open(unit=nvx1,file='t.dat')
         REWIND NVX1
c         READ(NVX1,1111) N
 1111    FORMAT(A56)
 1050    FORMAT(2A8)
         IF(IOG1.NE.0) READ(NVX1,1060) B3,G1
 1060    FORMAT(A8,A5)
         READ(NVX1,1000)(B(I),V1(I),XA1(I),I=1,N1)
 1000    FORMAT(A8,A4,A5)
         DO 30 I=1,N2
         IOV(I)=0
         READ(NVX1,1010) BA(I),BB(I),X1,IMVA(I)
 1010    FORMAT(2A8,A4,I5)
         IF(X1.EQ.'OV 1'.OR.X1.EQ.'OV 2') IOV(I)=1
 30      CONTINUE
         CLOSE(UNIT=NVX1)
         OPEN(UNIT=NVX2,FILE='IN.DAT')
         READ(NVX2,*) ixc,iyc
         READ(NVX2,1020) IHX1,IHY1,IHX2,IHY2
 1020    FORMAT(4I3)
         READ(NVX2,1030)(TYPE(I),IX1(I),IY1(I),IW(I),IVX1(I),IVY1(I)
        1,INX1(I),INY1(I),IBX1(I),IBY1(I),IG1(I),IRX1(I),IRY1(I),I=1,N1)
 1030    FORMAT(A1,9I3,I1,2I3)
```

```
      READ(NVX2,1040)(IX2(I),IY2(I),IX3(I),IY3(I),IX4(I),IY4(I)
     1,IX5(I),IY5(I),I=1,N2)
 1040 FORMAT(8I3)
      read(nvx2,*)ncb
      read(nvx2,3355)(ccb(i),icx(i),icy(i),i=1,ncb)
3355  format(a3,2i4)
      read(nvx2,*)ntr
      read(nvx2,7777)(trna(i),itx(i),ity(i),i=1,ntr)
7777  format(a2,2i4)
      REWIND NVX2
      CLOSE(UNIT=NVX2)
C ******************************************
      do111i=1,n1
      ix1(i)=ix1(i)-ixc
      ivx1(i)=ivx1(i)-ixc




      inx1(i)=inx1(i)-ixc
      ibx1(i)=ibx1(i)-ixc
      irx1(i)=irx1(i)-ixc
      iy1(i)=iy1(i)-iyc
      ivy1(i)=ivy1(i)-iyc
      iny1(i)=iny1(i)-iyc
      iby1(i)=iby1(i)-iyc
      iry1(i)=iry1(i)-iyc
111   continue
      do112i=1,n2
      ix2(i)=ix2(i)-ixc
      ix3(i)=ix3(i)-ixc
      ix4(i)=ix4(i)-ixc
      ix5(i)=ix5(i)-ixc
      iy2(i)=iy2(i)-iyc
      iy3(i)=iy3(i)-iyc
      iy4(i)=iy4(i)-iyc
      iy5(i)=iy5(i)-iyc
112   continue
      do113i=1,ncb
      icx(i)=icx(i)-ixc
```

```
      READ(NVX2,1040)(IX2(I),IY2(I),IX3(I),IY3(I),IX4(I),IY4(I)
     1,IX5(I),IY5(I),I=1,N2)
 1040 FORMAT(8I3)
      read(nvx2,*)ncb
      read(nvx2,3355)(ccb(i),icx(i),icy(i),i=1,ncb)
3355  format(a3,2i4)
      read(nvx2,*)ntr
      read(nvx2,7777)(trna(i),itx(i),ity(i),i=1,ntr)
7777  format(a2,2i4)
      REWIND NVX2
      CLOSE(UNIT=NVX2)
C ********************************************
      do111i=1,n1
      ix1(i)=ix1(i)-ixc
      ivx1(i)=ivx1(i)-ixc




      inx1(i)=inx1(i)-ixc
      ibx1(i)=ibx1(i)-ixc
      irx1(i)=irx1(i)-ixc
      iy1(i)=iy1(i)-iyc
      ivy1(i)=ivy1(i)-iyc
      iny1(i)=iny1(i)-iyc
      iby1(i)=iby1(i)-iyc
      iry1(i)=iry1(i)-iyc
111   continue
      do112i=1,n2
      ix2(i)=ix2(i)-ixc
      ix3(i)=ix3(i)-ixc
      ix4(i)=ix4(i)-ixc
      ix5(i)=ix5(i)-ixc
      iy2(i)=iy2(i)-iyc
      iy3(i)=iy3(i)-iyc
      iy4(i)=iy4(i)-iyc
      iy5(i)=iy5(i)-iyc
112   continue
      do113i=1,ncb
      icx(i)=icx(i)-ixc
```

```
      icy(i)=icy(i)-iyc
113   continue
      do114i=1,ntr
      itx(i)=itx(i)-ixc
      ity(i)=ity(i)-iyc
114   continue
c------------------------------------------------------------------------------
      CALL GRAF
      CALL HEAD
      DO 40 I=1,N1
      CALL DBUS(B(I),VI(I),XAI(I),IX1(I),IY1(I),IW(I),IVX1(I),IVY1(I)
     1,INX1(I),INY1(I),IBX1(I),IBY1(I),TYPE(I))
      IF(IGI(I).EQ.1)CALL GENER(IX1(I),IY1(I),TYPE(I),IRX1(I),IRY1(I))
      if(igl(i).eq.2) call load(ix1(i),iy1(i),irx1(i),iry1(i))
 40   CONTINUE
      DO 50 I=1,N2
 50   CALL DLIN(IMVA(I),IX2(I),IY2(I),IX3(I),IY3(I),IX4(I),IY4(I)
     1,IX5(I),IY5(I))
      do71i=1,ncb
      icx1=icx(i)
      icy1=icy(i)
      cb=ccb(i)
      call cktb(icx1,icy1,cb)
71    continue
      do81i=1,ntr
      call tran(itx(i),ity(i),trna(i))
81    continue
      nct=4
      call textf(170,1,nct

      CALL IMOD
c--------------------------------------------------------
      WRITE(*,1127)
1127   format(t10,'GS1 ,t30,'GS2',T60,'CIRCUIT BREAKER STATUS'/
     1'GEN VOLT (rms)',T15,'(REF)',T25,'GEN VOLT (rms)',T45,'(REF)',
     2 T60,'CB1 ,T70,'CB2')
c---------------------------------------------
      stop
      END
      SUBROUTINE GRAF
      EXTERNAL GMODE,GPAGE,DISP,LEVEL,CLRSCR






      DATA IONE/1/
```

```
 CALL GMODE
 CALL GPAGE(IONE)
 CALL DISP(IONE)
 CALL LEVEL(IONE)
 CALL CLRSCR
 RETURN
 END
 SUBROUTINE HEAD
 EXTERNAL TEXTF
 COMMON/SYS1/N1,N2,IOU1,IBUS,IBAR,NVX2,IHX1,IHY1,IHX2,IHY2,N,IOB1
1,IOG1,B1,B2,B3,G1
 CHARACTER *56 N
 CHARACTER *16 A1,A2
 CHARACTER *8 B1,B2,B3
 CHARACTER *5 D,G1
 CHARACTER *4 M
 CHARACTER *13 A3,A4
 DATA A1/'LOAD FLOW STUDY  '/,A2/'    OUTAGE STUDY  '/
 DATA A3/'    BRANCH OUT '/,A4/'GENERATOR OUT '/
 DATA D/'DROP= '/,M/'M.W. '/
 IF1=16
 IF2=56
 IF3=8
 IF4=5
 IF5=13
 IF6=4
 IX3=IHX2+153
 IX4=IHX2+279
 IX5=IHX2+360
 IX6=IHX2+414
 IX7=IHX2+468
 CALL TEXTF(IHX1,IHY1,IF2,N)
 RETURN
 END
 SUBROUTINE DBUS(B,V,A,IX1,IY1,IW,IVX1,IVY1,INX1,INY1,IBX1
1,IBY1,TYPE)
 CHARACTER *8 B
 CHARACTER *4 V
 CHARACTER *5 A
 EXTERNAL BLKFIL,TEXTF
 I5=3
 I6=4
 I7=8
 I8=5
 I1=IX1-IW/2
 I2=IY1+1
```

```
      I3=IX1-1
      I4=IX1+IW/2
      IF(TYPE.EQ.'H') CALL BLKFIL(I1,I2,IW,I5)
      IF(TYPE.EQ.'V') CALL BLKFIL(I3,I4,I5,IW)
      CALL TEXTF(IBX1,IBY1,I7,B)
c      CALL TEXTF(IVX1,IVY1,I6,V)
c      CALL TEXTF(INX1,INY1,I8,A)
      RETURN
      END
      SUBROUTINE DLIN(IMVA,IX2,IY2,IX3,IY3,IX4,IY4,IX5,IY5)
      EXTERNAL PUTPT,DLINE
      CHARACTER * 8 BA,BB
      IF(IMVA.EQ.0) RETURN
      CALL PUTPT(IX2,IY2)
      CALL DLINE(IX3,IY3)
```

```
      CALL DLINE(IX4,IY4)
      CALL DLINE(IX5,IY5)
      RETURN
      END
      SUBROUTINE TMOD
      EXTERNAL HRDCPY,CLSCR,TMODE,TEXTF
      CHARACTER *20 A,B
      DATA A/'1. HARDCOPY 2. EXIT  '/
      DATA B/'                     '/
      DATA I2/2/
      DATA I1/1/
      DATA TWO/'2'/
      IX=1
      IY=340
      I=20
c      CALL TEXTF(IX,IY,I,A)
      READ(*,*) N
      GO TO (10,20),N
c 10   CALL TEXTF(IX,IY,I,B)
10      CALL HRDCPY(TWO)
c 20   CALL CLRSCR
```

```
20        CALL TMODE
      RETURN
      END
      SUBROUTINE BLINK(IX2,IY2,IX3,IY3,IX4,IY4,IX5,IY5)
      EXTERNAL LEVEL,DLINE,PUTPT
      I1=1
      I2=2
      CALL LEVEL(I2)
      DO 10 I=1,200
      CALL PUTPT(IX2,IY2)
      CALL DLINE(IX3,IY3)
      CALL DLINE(IX4,IY4)
      CALL DLINE(IX5,IY5)
10    CONTINUE
      CALL LEVEL(I1)
      RETURN
      END
      SUBROUTINE GENER(IX1,IY1,TYPE,IRX1,IRY1)
      EXTERNAL PUTPT,CIRC,DLINE,LEVEL,TEXTF
      CHARACTER *1 G
      DATA G/ G /
      I1=1
      I2=2
      IRAD=15
      IR1=IRAD*2/3
      IR3=IRX1-4
      IR4=IRY1+3
      IF (TYPE.EQ. V ) GO TO 10
      CALL PUTPT(IRX1,IY1)
      CALL DLINE(IRX1,IRY1)
      CALL CIRC(IRX1,IRY1,IRAD)
      CALL PUTPT(IRX1,IRY1)
      CALL LEVEL(I2)
      IF (IRY1.GT.IY1) THEN
             IT1=IY1+8
             IR2=IRY1-IR1
      ELSE
             IR2=IRY1+IR1
             IT1=IY1-8
      ENDIF
      CALL DLINE(IRX1,IR2)




      CALL LEVEL(I1)
```

```
      CALL TEXTF(IR3,IR4,I1,G)
      RETURN
 10   CALL PUTPT(IX1,IRY1)
      CALL DLINE(IRX1,IRY1)
      CALL CIRC(IRX1,IRY1,IRAD)
      CALL PUTPT(IRX1,IRY2)
      CALL LEVEL(I2)
      IF(IRX1.GT.IX1) THEN
            IR2=IRX1-IRAD
      ELSE
            IR2=IRX1+IRAD
      ENDIF
      CALL DLINE(IR2,IRY2)
      CALL LEVEL(I1)
      CALL TEXTF(IR3,IR4,I1,G)
      RETURN
      END
      SUBROUTINE CLS
      EXTERNAL TMODE,GMODE,CLRSCR
      CALL GMODE
      CALL CLRSCR
      CALL TMODE
      RETURN
      END
C-------------------------------------------------------------
      SUBROUTINE cktb(IX1,IY1,cb)
      EXTERNAL PUTPT,CIRC,DLINE,LEVEL,TEXTF
      CHARACTER *3 cb
      icx1=ix1-3
      icy1=iy1-3
      call putpt(icx1,icy1)
      icx2=ix1+3
      icy2=iy1+3
      call dline(icx2,icy2)
      icy1=iy1-3
      call putpt(icx2,icy1)
      icx2=ix1-3
      icy2=iy1+3
      call dline(icx2,icy2)
      call textf(ix1+4,iy1+2,3,cb)
      return
      end
C-------------------------------------------------------------
      SUBROUTINE tran(IX1,IY1,tr)
      EXTERNAL PUTPT,CIRC,DLINE,LEVEL,TEXTF
      CHARACTER *2 t
```

4

```
      call putpt(ix1,iy1)
      call level(0)
      call dline(ix1,iy1+2)
      call level(1)
      call dline(ix1-3,iy1-3)
      call dline(ix1-6,iy1)
      call putpt(ix1,iy1)
      call dline(ix1+3,iy1-3)
      call dline(ix1+6,iy1)
      iy1=iy1+2
      call putpt(ix1,iy1)
      call dline(ix1-3,iy1+3)
      call dline(ix1-6,iy1)
      call putpt(ix1,iy1)
      call dline(ix1+3,iy1+3)
```

```
      call dline(ix1+6,iy1)
      call textf(ix1+7,iy1,2,tr)
      return
      end
C---------------------------------------------------------------------
      SUBROUTINE load(IX1,IY1,irx1,iry1)
      EXTERNAL PUTPT,CIRC,DLINE,LEVEL,TEXTF
      CHARACTER *1 LD
      DATA LD/'L'/
      CALL PUTPT(IRX1,IY1)
      CALL DLINE(IRX1,IRY1)
      if(iy1.gt.iry1) then
        call dline(irx1-4,iry1+4)
        call putpt(irx1,iry1)
        call dline(irx1+4,iry1+4)
      else
        call dline(irx1-4,iry1-4)
        call putpt(irx1,iry1)
        call dline(irx1+4,iry1-4)
      endif
      call textf(irx1+6,iry1+6,1,LD)

      return
      end
```

APPENDIX-C

```
        ORG     16384
        EXTRN   TEMP02,MEM02,MEM03,TEMP01,STORE2,STORE3,MEMSUM1,SUM1
        EXTRE   COUNTER,ENTRY,STORE4
        EXTRN   MEM4,MBINTA,TYPE,MEM17,BINTA,DISP1,DISP2,DISP3
        EXTRN   DISP4,READ1,READ0
        EXTRN   WRONG,MEM00,STORE0,STORE1,TEMP00,SUM0,MEM01,MEMSUM0
        ;input from MEM00,TEMP00,STORE0
        ;output in MEM01,STORE1
        ;This routine converts a block of HEX BYTES TO its ASCII equivalent
HEXTA   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save all registers
        LHLD    STORE0  ;initialise the memory pointer from where HEX BLOCK
                        ; starts
        XCHG
        LHLD    STORE1  ;initialise the memory pointer for storing ASCII no
        XCHG
        LDA     TEMP00  ;load the number of bytes for conversion
        MOV     B,A     ;store in register B
DO1     MOV     A,M     ;get no. in accumulator
        ANI     0F0H
        RRC
        RRC
        RRC
        RRC
        ORI     30H     ;convert higher nibble to ASCII
        XCHG
        MOV     M,A     ;store it in memory
        INX     H
        XCHG
        MOV     A,M
        ANI     0FH
        ORI     30H     ;convert lower nibble to ASCII
        INX     H
        XCHG
        MOV     M,A     ;store its value
        INX     H
        XCHG
        DCR     B       ;check if conversion is over for all bytes
        JNZ     DO1     ;if not loop to DO1
        POP     PSW     ;if yes retrive registers
        POP     D
        POP     B
        POP     H
        RET             ;return
```

```
        ORG     16384
        EXTRN   TEMP02,MEM02,MEM03,TEMP01,STORE2,STORE3,MEMSUM1,SUM1
        EXTRN   COUNTER,ENTRY,STORE4
        EXTRN   MEM4,MBINTA,TYPE,MEM17,BINTA,DISP1,DISP2,DISP3,DISP4,READ1
        EXTRN   READ0
        EXTRN   WRONG,MEM00,STORE0,STORE1,TEMP00,SUM0,MEM01,MEMSUM0
        ;input from MEM00,TEMP00,STORE0
        ;output in MEM01,STORE1
        ;this routine converts a block of HEX BYTES TO its ASCII equivalent
HEXTA   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save all registers
        LHLD    STORE0  ;initialise the memory pointer from where
                        ;HEX BLOCK starts
        XCHG
        LHLD    STORE1  ;initialise the memory pointer for storing ASCII no.
        XCHG
        LDA     TEMP00  ;load the number of bytes for conversion
        MOV     B,A     ;store in register B
DO1     MOV     A,M     ;get no. in accumulator
        ANI     0F0H
        RRC
        RRC
        RRC
        RRC
        ORI     30H     ;convert higher nibble to ASCII
        XCHG
        MOV     M,A     ;store it in memory
        INX     H
        XCHG
        MOV     A,M
        ANI     0FH
        ORI     30H     ;convert lower nibble to ASCII
        INX     H
        XCHG
        MOV     M,A     ;store its value
        INX     H
        XCHG
        DCR     B       ;check if conversion is over for all bytes
        JNZ     DO1     ;if not loop to DO1
        POP     PSW     ;if yes retrive registers
        POP     D
        POP     B
        POP     H
        RET             ;return
        ;input from MEM00,TEMP00
        ;output in MEMSUM0,SUM0
```

```
        ;this routine calculates the checksum, stores and converts it
        ;into its ASCII equivalent
CHKSUM  PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;saves  the registers
        LHLD    STORE0  ;initiathe memory pointer from where the
                        ; HEX block starts
        LDA     TEMP00  ;load the no. of bytes in the block in accumulator
        MOV     B,A     ;and save in B
        XRA     A       ;clear accumulator
DO2     MOV     C,M
        ADD     C
        INX     H


        DCR     B
        JNZ     DO2     ;add all the bytes of string
        CMA             ;complement to get the checksum
        MOV     C,A
        STA     TEMP01  ;store the checksum
        LHLD    SUM0    ;initialise memory pointer in which ASCII equivalent
        ANI     0F0H    ;of checksum will be stored
        RRC
        RRC
        RRC
        RRC
        ORI     30H     ;convert higher nibble to ASCII and store
        MOV     M,A
        INX     H
        MOV     A,C
        ANI     0FH
        ORI     30H     ;convert lower nibble to ASCII and store
        MOV     M,A
        POP     PSW
        POP     D
        POP     B
        POP     H       ;retrive all registers
        RET             ;return
        ;input from meml,sto
```

```
        ;output to the 8251
COMMON  MACRO
        IN      0F1H
        ANI     01H
        ENDM
ADDHEAD PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW        ;save all registers
        LHLD    STORE1
        LDA     TEMP00     ;initialise the memory pointer from where the ASCII
        ADD     A          ;block is stored
        MOV     B,A        ;load no. of bytes in register C
DO3     COMMON
        JZ      DO3
        MVI     A,02H
        OUT     0F0H       ;transmit STX
DO4     COMMON
        JZ      DO4
        MOV     A,M
        OUT     0F0H
        INX     H
        DCR     B
        JNZ     DO4        ;transmit the ASCII block
DO5     COMMON
        JZ      DO5
        MVI     A,03H
        OUT     0F0H       ;transmit ETX
        LHLD    SUM0
        MVI     B,02H      ;initialise the memory pointer  where
                           ; checksum is stored
DO6     COMMON
        JZ      DO6
        MOV     A,M
        OUT     0F0H
        INX     H
        DCR     B
        JNZ     DO6        ;transmit two bytes of checksum




DO7     COMMON
```

```
        MVI     A,04H    ;transmit EOT
        OUT     0F0H
        POP     PSW
        POP     D
        POP     B
        POP     H        ;restore all registers
        RET              ;return
        ;input from MEM02,STORE2,SUM1,STORE3
        ;output in MEM03,MEMSUM1,TEMP02
        ;this routine converts a block of ASCII no. to its HEX equivalent
ASTH    PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW      ;save registers
        MVI     C,0      ;clear counter
        LHLD    STORE2   ;initialise the memory pointer to the starting addres
        XCHG             ;of the block
        LHLD    STORE3   ;initialise the mempry pointer from where to store
        XCHG
        INX     H
DO8     MOV     A,M
        SUI     30H
        RLC
        RLC
        RLC
        RLC
        MOV     B,A      ;take the first byte and get the higher nibble in C
        INX     H
        MOV     A,M
        SUI     30H      ;take the second byte and get the lower nibble
        ADD     B        ;add to get HEX number
        XCHG
        MOV     M,A
        INR     C        ;store in memory
        MOV     A,C
        STA     TEMP02   ;store the no. of bytes
        INX     H
        XCHG
        INX     H
        MOV     A,M      ;all bytes over ? check ETX
        CPI     03H      ;ETX ?
        JNZ     DO8      ;NO : loop to DO8
        INX     H
        XCHG
        LHLD    SUM1     ;initialise the memory pointer from where
                         ; to store checksum
        XCHG
```

```
        MOV     A,M
        SUI     30H
        RLC
        RLC
        RLC
        RLC
        MOV     B,A     ;get first byte and store higher nibble in B  registe
        INX     H
        MOV     A,M
        SUI     30H     ;get second byte and convert it  into lower nibble
        ADD     B       ;add to get byte
        XCHG
        MOV     M,A     store checksum
        POP     PSW



        POP     B
        POP     H       ;restore registers
        RET             ;return
        ;this routine checks for SOH
        ;it is for  entry STEP-I and checks if SOH is recieved from MCS
ENTRY1  PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
DO23    IN      0F1H
        ANI     02H
        JZ      DO23    ;check  for reciever ready
        IN      0F0H    ;input the byte  and  save in register B
        MOV     B,A
        ANI     08H     ;check for error
        JNZ     DO9     ;return on error
        MOV     A,B
        CPI     01H     ;compare with SOH on no error
        JNZ     DO9     ;NO : jump to return  back
        MVI     A,0     ;YES : load counter with  0 bytes and
                        ;  initialise  entry
        STA     COUNTER ;entry step for next interrupt is 02
        MVI     A,02H
        STA     ENTRY
        MVI     A,06
DO10    IN      0F1H
        ANI     01H
        JZ      DO10    ;send ACK
```

```
        OUT     0F0H
DD5     POP     PSW
        POP     D
        POP     B
        POP     H           ;restore registers
        HLT
        ;this routine tests the validity of TYPE of data communication
ENTRY2  PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW         ;save registers
DD22    IN      0F1H
        ANI     02H         ;check for reciever ready
        JZ      DD22        ;wait till reciever ready
        IN      0F0H        ;read byte
        STA     TYPE
        ANI     08H         ;check for  error
        JNZ     NO          ;jump to NO on error
        LDA     TYPE
        CPI     30H
        JNZ     STEP1
F1      IN      0F1H
        ANI     01H
        JZ      F1
        MVI     A,06H
        CALL    ENTRY3
        JMP     LAST
        OUT     0F0H
        CALL    ENTRY3      ;check if type is 0
        JMP     LAST        ;yes : send ACK and call entry STEP3 and jump to LAST
STEP1   CPI     31H
        JZ      STEP2
        CPI     32H


        JZ      STEP2
```

```
        CPI     33H
        JZ      STEP2
        CPI     34H
        JZ      STEP2
        CPI     35H
        JZ      STEP2
        CPI     36H
        JZ      STEP2
        CPI     37H
        JZ      STEP2
        CPI     38H
        JZ      STEP2
        CPI     39H
        JZ      STEP2
        CPI     41H
        JZ      STEP2
        CPI     42H      ;check for valid code
        JZ      STEP2    ;jump to STEP2
        JMP     NO       ;No valid code, increment the count and return by
                         ;jumping to NO
STEP2   MVI     A,02H
        STA     ENTRY
        JMP     YES
NO      LDA     COUNTER
        INR     A
        STA     COUNTER  ;increment counter
        CPI     03H      ;check if counter is 3
        JZ      DO11     ;if counter = 3 initialise  entry step 1
        MVI     A,02H    ;No:entry step remains two, jump to LAST
        STA     ENTRY
        JMP     LAST
DO11    MVI     A,01H
        STA     ENTRY
        JMP     LAST     ;initialise entry step 1 and jump to LAST
YES     MVI     A,06H
DO12    IN      0F1H
        ANI     01H
        JZ      DO12
        OUT     0F0H     ;send  ACK
LAST    POP     PSW
        POP     D
        POP     B
        POP     H        ;restore
```

```
                                        registers
        HLT
        ;this routine sends a preassambled packet
        ;BEFORE entering this routine SET COUNTER to 00
ENTRY3  PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW     ;save registers
DO16    CALL    ADDHEAD
        CALL    DELAY
DO24    IN      0F1H
        ANI     02H
        JZ      DO24
        IN      0F0H    ;recieve  byte  if reciever  ready else loop
        MOV     B,A
        CPI     06H
        JZ      DO13    ;jump to

                                        DO13 if  ACK recieved
        LDA     COUNTER
        INR     A

        STA     COUNTER ;increment counter if ACK not recieved
        CPI     03H
        JZ      DO13    ;if counter is 3 junp to DO13
        JMP     DO16    ;else send packet again
DO13    MVI     A,01H
        STA     ENTRY   ;set entery step
        POP     PSW
        POP     D
        POP     B
        POP     H       ;restore registers
        HLT
        ;this routine tests the validity of data received
        ;BEFORE entering this routine SET COUNTER to 00
ENTRY4  PUSH    H
        PUSH    B
        PUSH    D
```

```
        PUSH    PSW     ;save registers
        LHLD    STORE2
DO14    IN      0F1H
        ANI     02H
        JZ      DO14
        IN      0F0H    ;recieve byte if  reciever  ready else loop back
        MOV     B,A
        CPI     04H     ;ETX ?
        JZ      DO15    ;yes : jump to DO15
        MOV     M,B     ;store this byte
        INX     H
        JMP     DO14    ;no : recieve next byte
DO15    CALL    CHECK
        POP     PSW
        POP     D
        POP     B
        POP     H       ;restore registers
        HLT
DELAY   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
        LXI     B,0     ;load count
DO17    DCX     B
        MOV     A,C
        ORA     B       ;check if count is 0
        JNZ     DO17    ;no : loop again
        POP     PSW     ;yes
        POP     D
        POP     B
        POP     H       ;restore registers
        RET
CHECK   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
        CALL    ASTH    ;convert ASCII to ots equivalent HEX
        LDA     TEMP02
        STA     TEMP00
        LHLD    STORE3
        SHLD    STORE0  ;initialise memory to calculate checksum
        CALL    CHKSUM  ;calculate checksum
        LHLD    SUM1
        MOV     B,M
        LDA     TEMP01
        CMP     B       ;correct data recieved ?



        JNZ     SENDNAK ;no : send NACK
```

```
        LDA     TYPE      ;yes
        CPI     31H
        JNZ     STEP3
        CALL    SUB31
        JMP     DO18
STEP3   CPI     32H
        JNZ     STEP4
        CALL    SUB32
        JMP     DO18
STEP4   CPI     33H
        JNZ     STEP5
        CALL    SUB33
        JMP     DO18
STEP5   CPI     34H
        JNZ     STEP6
        CALL    SUB34
        JMP     DO18
STEP6   CPI     35H
        JNZ     STEP7
        CALL    SUB35
        JMP     DO18
STEP7   CPI     36H
        JNZ     STEP8
        CALL    SUB36
        JMP     DO18
STEP8   CPI     37H
        JNZ     STEP9
        CALL    SUB37
        JMP     DO18
STEP9   CPI     38H
        JNZ     STEP10
        CALL    SUB38
        JMP     DO18
STEP10  CPI     39H
        JNZ     STEP11
        CALL    SUB39
        JMP     DO18
STEP11  CPI     41H
        JNZ     STEP12
        CALL    SUB41
        JMP     DO18
STEP12  CALL    SUB42     ;check which code is recieved  and call its subroutine
DO18    MVI     A,01H
        STA     ENTRY     ;initialise entry step to 1
        MVI     A,00
        STA     SET       ;load memory set with zero count
```

```
DO19    IN      0F1H
        ANI     01H
        JZ      DO19
        MVI     A,06H
        OUT     0F0H    ;send ACK
        JMP     DO21    ;jump to return
SENDNAK LDA     COUNTER
        INR     A
        STA     COUNTER
        CPI     03H     ;increment counter  and  check if = 3
        JZ      DO20    ;yes : jump to DO20
        MVI     A,04H
        STA     ENTRY   ;no : make entry step 4
        MVI     A,FF
        STA     SET     ;load memory with FF Hex
```

```
DO50    IN      0F1H
        ANI     01H
        JZ      DO20
        MVI     A,15H
        OUT     0F0H    ;send NAK
        MVI     A,04H
        STA     ENTRY   ;make  entry step to 4
DO21    POP     PSW
        POP     D
        POP     B
        POP     H       ;restore registers
        RET
DO20    MVI     A,00
        STA     SET
        MVI     A,01
        STA     ENTRY   ;set entry step  to 1 and memory set to 0
        JMP     DO50    ;jump to DO50 to send NAK
        ;this subroutine is for TYPE 1 which prompts to start
                        ;a particular plant
SUB31   PUSH    H
        PUSH    B
```

```
        PUSH    D
        PUSH    PSW     ;save registers
        LHLD    STORE3
        MOV     A,M
        STA     MEM17   ;load the plant number  and store in MEM17
        CALL    DISP2   ;call DISPLAY to display start plant number
        POP     PSW
        POP     D
        POP     B
        PPOP     H      ;restore registers
        RET             ;return
SUB32   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
        LHLD    STORE3
        MOV     A,M
        STA     MEM17   ;load plant number in MEM17
        CALL    DISP4   ;call display to display "Stop Plant No."
        POP     PSW
        POP     D
        POP     B
        POP     H       ;restore registers
        RET             ;return
SUB33   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
        LHLD    STORE3
        MOV     A,M
        STA     MEM17
        CALL    DISP1   ;start switch device number
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
SUB34   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW




        LHLD    STORE3
```

```
        MOV     A,M
        STA     MEM17
        CALL    DISP3   ;stop switch device number
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
SUB36   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
        LHLD    STORE3
        MOV     A,M     ;load the variable number in accumulator
        SBI     05H
        ADD     A
        MOV     B,A     ;calculate the offset to be added in  address
        INX     H       ;and save in B register
        MOV     D,M
        INX     H
        MOV     E,M     ;save the max. value in DE pair
        XCHG
        LXI     H,9F00H
        MOV     A,L
        ADD     B
        MOV     L,A
        MOV     M,E
        INX     H
        MOV     M,D     ;store the number in memory location
        POP     H
        POP     D
        POP     B
        POP     PSW
        RET
SUB37   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW     ;save registers
        LHLD    STORE3
        MOV     A,M
        SBI     05H
        ADD     A
        MOV     B,A     ;load the variable number and make an offset
        INX     H       ;and store in  register C
        MOV     D,M
        INX     H
```

```
        MOV     E,M       ;get the max. val. in DE reg. pair
        XCHG
        LXI     H,9F10H
        MOV     A,L
        ADD     B         ;set the memory pointer
        MOV     L,A
        MOV     M,E
        INX     H
        MOV     M,D       ;store the value in memory
        POP     H
        POP     D
        POP     B
        POP     PSW       ;restore  registers
        RET
SUB35   PUSH    H
```

```
        PUSH    B
        PUSH    D
        PUSH    PSW       ;save registers
        LHLD    STORE3
        MOV     A,M       ;load the variable no. in Accumulator
        CPI     01H
        JZ      DO25
        CPI     02H
        JZ      DO26
        CPI     03H       ;compare the no. and jump to its relevant location
        JZ      DO27
        LXI     H,0B070H
        SHLD    MBINTA    ;set the memory pointer for 04 variable
                          ; and jump to DO28
        JMP     DO28
DO25    LXI     H,0B05EH
        SHLD    MBINTA    ;Bset the memory pointer for 01 variable
                          ;and jump to DO28
        JMP     DO28
DO26    LXI
```

```
                H,08064H
        SHLD    MBINTA    ;set the memory pointer for 02 variable
                          ;and jump to D028
        JMP     D028
D027    LXI     H,0806AH
        SHLD    MBINTA    ;set the memory pointer for 03 variable
                          ;and jump to D028
D028    LHLD    STORE3
        INX     H
        MOV     D,M
        INX     H
        MOV     E,M       ;load the reference variable in DE reg. pair
        XCHG
        SHLD    MEM4      ;bring it in HL pair and store HL in MEM4 to convert
                          ; it
        CALL    BINIA     ;into ASCII
        POP     H
        POP     D
        POP     B
        POP     PSW
        RET
SUB38   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW       ;save registers
        MVI     A,0FFH
        OUT     58H
        MVI     A,0FFH
        OUT     58H       ;initialise  the  counter 0 to FFFFH
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
SUB39   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        MVI     A,0FFH
        OUT     59H
        MVI     A,0FFH
        OUT     59H       ;initialise the counter to  FFFFH
        POP     PSW




        POP     D
```

```
        POP     B
        POP     H
        RET
READ00  PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        MVI     A,00H
        OUT     5BH
        IN      58H
        CMA
        MOV     L,A
        IN      58H
        CMA
        MOV     H,A
        SHLD    READ0   ;read counter 0 and store in READ0
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
READ01  PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        MVI     A,40H
        OUT     5BH
        IN      59H
        CMA
        MOV     L,A
        IN      59H
        CMA
        MOV     H,A
        SHLD    READ1   ;read counter 1 and store in READ0
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
        ;this routine  reads the value  of  counter 0  and sends
        ;that  value  to MCS
SUB41   PUSH    H
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    READ00  ;read the value of counter 0
```

```
        MVI     A,04H
        STA     TEMP00
        LXI     H,READ0
        SHLD    STORE0
        CALL    HEXTA   ;convert it into ASCII
        CALL    CHKSUM  ;call CHKSUM to calculate checksum
        CALL    ADDHEAD ;send packet to MCS
        POP     PSW
        POP     D
        POP     B
        POP     H       ;restore registers
        RET
        ;this routine reads the value of counter 1  and
        ;sends that value to MCS
SUB42   PUSH    H
```

```
        PUSH    B
        PUSH    D
        PUSH    PSW
        CALL    READ01
        MVI     A,04H
        STA     TEMP00
        LXI     H,READ1
        SHLD    STORE0
        CALL    HEXTA
        CALL    CHKSUM
        CALL    ADDHEAD
        POP     PSW
        POP     D
        POP     B
        POP     H
        RET
```

END

```
code segment
assume cs:code, ds:code, es:nothing, ss:nothing
org 100H
start: jmp main
;------------------------data area----------------------
pkt_len                 DB        150
Tx_Buf                  DB        pkt_len dup(0)
T_buf                   DB        201pkt(0)
Type                    DB        8
KB_FLAG                 DB         ? ;0-read data from K.B.
Counter                 DB        0
Rx_Buf                  DB        pktlen dup(0)
R_Buf                   DB        201 dup(0)
ER_MSG                  DB        "Fatal error Existing $"
Rcd_ACK                 DB        0 ; 0-Ack_Rcd , 1-Ack_not_rcd
Nak_error               DB        ? ; 0-Nak_not_Rcd , 1-Nak_Rcd
ACK                     EQU       06h
S0H                     EQU       01h
Mx_RtRy                 EQU       -3
EOT                     EQU       04h
STX                     EQU       02h
ETX                     EQU       03h
;This procedure initializes the comport to 4800 baud, even parity, 1 Stop bi
INIT:               PROC     NEAR
                    MOV 9H,0
                    MOV AL, 0C7H
                    INT  14H
                    RET
INIT                ENDP
SEND_BYTE           PROC     NEAR
;Call with al=byte to be tranfered , In case of rerror byte
;is sent thrice
                    MOV CL,3          ;error count
S1:                 MOV AH,1
                    INT 14H           ;SEND BYTE IN AL
                    TEST AH,80H       ;COMPARE ,IF ERROR
                    JZ S2             ;IF NOT EXIT
                    DEC CL            ;DECREMENT COUNTER
                    JNZ S1            ;RETRY IF ERROR
S2:                 RET
SEND_BYTE:          ENDP
;This routine sends packet to RTU
SEND_PKT            PROC            NEAR
;This procedure sends the packet , using send_byte Procedure
                    MOV S1,OFFSET Tx_BUF     ;start of buffer
SEND_NEXT:          LODSB             ;Load byte in AL
                    CALL SEND_BYTE
                    LOOP SEND_NEXT   ;Repeat till the end of Buffer
                    R
```

```
                    ET
SEND_PKT:           ENDP
;This routine gets byte comport in AL, in case of error repeats three times
GET_BYTE            PROC                NEAR
                    MOV CL,3            ;Retry Count
G1:                 MOV AH,2
                    INT 14H
                    CMP AH,Ø            ;Compare for error
                    JZ G2               ;no error,exit
                    DEC CL              ; error
                    JNZ G1
G2:                 RET
GET_BYTE            ENDP
;to call,Es:di=Segment:offset of rec. buffer, CX=Packet Size
GET_PKT             PROC                NEAR
G1:                 MOV CX,Ø
                    CALL GET_BYTE       ;Receive byte
                    STOSB               ;Store it
                    INC CX
                    CMP AL,EOT          ;if eot
                    JZ G3               ;exit
                    JMP G1
G3:                 RET
GET_PKT             ENDP
                    MOV DI,OFFSET Rx_BUF
;no need of packet size
MAKE_PKT            PROC                NEAR
                    MOV SI,OFFSET IBUFFER
                    PUSH CX
                    MOV DI,OFFSET Tx_BUFFER
                    MOV AL ,STX
                    STOSB
M1:                 LODSB
                    CALL HEX_to_ASCII ;Convert to numeric ASCII
                    STOSW
                    LOOPNZ M1
                    MOV AL,ETX
                    STOSB
                    MOV SI,OFFSET T_Buf
                    INC S1              ;Leave STX
                    POP CX
                    CALL CHK_SUM        ;calculate check sum
                    MOV AL,CHK_SUM
                    CALL HEX_to_ASCII ;convert cs to ASCII
                    STOSW               ;save it
                    MOV AL,EOT
                    STOSB
                    RET
MAKE
```

```
    _PKT          ENDP
;before calling the routine load CX with no. of HEX data bytes
;and load dx with the no. of ASCII bits
HEX_to_ASCII    PROC            NEAR
;This routine converts a byte to numeric ASCII word
;That is 3C is 33 3C . Call with AL=bytes to convert
                PUSH CX
                PUSH AX             ;save registers
                AND AL,ØFH          ;mask higher nibble
                ADD AL,3ØH          ;convert into ASCII
                POP DX              ;get no. in DL
                AND DL,OFOH         ;mask lower nibble
                MOV CL,4
                SHR DL,CL           ;Rotate right the byte 4 times
                ADD DL,3ØH          ;convert into ASCII
                MOV AH,DL           ;get the higher byte in AH
                POP CX              ;Pop registers
                RET                 ;returns code AX
HEX_to_ASCII    ENDP
;this routine converts the numeric ASCII word in AX to Hex byte in AL
ASCII_to_HEX    PROC NEARR
                SUB AL,3ØH          ;convert into lower nibble
                SUB AH,3ØH          ;get higher 4 bits in AH
                MOV CL,4
                SHL AH,CL
                OR AL,AH            ;get byte in AL
                RET
ASCII_to_HEX    ENDP
;this routine calculates the checksum and returns with check sum in AL
CHK_SUM         PROC NEAR
                XOR AX,AX           ;AX=Ø
C1:             LODSB               ;get byte in AL
                XOR AH,AL           ;add to partial result
                LOOPNZ C1           ;if bytes over , no:loop1
                MOV AL,AH           ;yes :get sum in AL
                NEG AL              ;convert into checksum
CHK_SUM         ENDP
;this routine makes the packet sends the packet to comm port and ckecks
;if ACK recd. or not. In case ACK not recd. , packet is sent twice
SEND_OUT        PROC NEAR
                PUSH CX             ;save no. of hex bytes
                CALL MAKE_PKT       ;make packet
                POP CX              ;get hex data bytes
Ø1:             MOV RECD_ACK,Ø
                SHL CX,1
                ADD CX,6            ;get packet size in cx
                CALL WAIT15         ;15 ms wait
                CMP RECD_AC
```

```
                CMP RECD_ACK,1  ;is ack recd.?
                JNZ Ø2          ;NO error exit
                INC COUNTER     ;else increment counter
                CMP COUNTER,MAX_RETRY
                JP Ø1           ;not exceeding, so go back
                MOV ERROR, 1    ; exceeds max retry, set error flag
Ø2:             RET
SEND_OUT        ENDP
;this routine recieves the packet from RTU and checks its validity. If
;valid, sends ACK else sends NAK
GET_IN          PROC NEAR
                MOV DI,OFFSET Rx_BUFFER ; initialise DI with offset Rx_buffer
                PUSH DI         ;save offset
                CALL GET_PKT    ;recieve packet
                PUSH CX         ;save packet length
                POP SI          ;get starting address of block in SI
                MOV DI,OFFSET R_BUFFER  ; initialise DI with offset R_buffer
                INC SI
                SUB CX,5        ;get data length in CX
G1:             LODSW
                CALL ASCII_TO_HEX
                STOSB           ;convert the data into hex
                LOOPNZ  G1
                PUSH SI
                MOV SI,OFFSET R_BUFFER  ;initialise SI with adress of checksum
                CALL CHK_SUM    ;calculate checksum
                CALL HEX_TO_ASCII; convert checksum into ASCII
                POP SI
                MOV DL, [SI]
                INC SI
                MOV DH,[SI]
                CMP AX,DX       ;compare two checksums for validity of packet
                JZ SAME         ;valid, jump to save
                MOV AL,1
                MOV NAK_ERROR,1 ;no, set nak_error flag to 1
                MOV DI,OFSET Rx_BUFFER
                POP CX
                MOV AX,Ø
FLUSH:          STOSW
                LOOP FLUSH      ;clear the buffer
                MOV AL, NAK
                CALL SEND_BYTE  ;send NAK
                JMP OT
SAME:           MOV AL,ACK      ;send ACK
                CALL SEND_BYTE
                MOV AL,Ø
                MOV NAK_ERROR,1
OT:             RET             ;return
GET_IN          ENDP
```

```
        PUBLIC  _DISPLAY
MAIN    PUSH    CS
        POP     DS
        PUSH    CS
        POP     ES                  ;initalise all segments to same value
W1:     MOV     CL,3                ;set the counter to 3
W5:     MOV     AH,1
        MOV     AL,SOH              ;transmit SOH
        INT     14H
        CALL    WAIT15              ;wait for 15 ms
        MOV     AH,2
        INT     14H                 ;read if acknowledgement recieved
        CMP     CL,ACK              ;is ACK recd?
        JZ      W4                  ;YES,jump to read type from kbd.
        DEC     CL                  ;NO, decrement the counter
        JNZ     W5                  ;if counter zero NO, send SOH  again
        JMP     PRCS_ERROR          ;YES, jump to process error
W4:     MOV     AH,1
        INT     16H                 ;check keyboard for any key pressed
        JZ      W1
        MOV     AH,Ø
        INT     16H                 ;read kbd.
        CMP     AL,42H
        JG      ERROR               ;is key pressed within our codes
        CMP     AL,3ØH              ; NO, jump to error routine
        JL      ERROR
        MOV     DI,OFFSET T_BUF ;YES , initialise DI with offset trans. buffer
        CMP     AL,38H              ;is byte 38H
        JZ      ZERO BYTE           ;YES, jump to send zero byte
        CMP     AL,39H              ;NO, is byte 39H
        JZ      ZERO BYTE           ;YES jump to send zero byte
        CMP     AL,35H              ;NO is byte 35H
        JZ      FIVE BYTES          ;YES jump to send five bytes
        CMP     AL,36H              ;NO is byte 36H
        JZ      FIVE BYTES          ;YES jump to send five bytes
        CMP     AL,37H              ;NO is byte 37H
        JZ      FIVE BYTES          ;YES jump to send five bytes
        CMP     AL,3ØH              ;NO is byte 3ØH
        JZ      GET_PACKET          ;YES get packet from RTU
        CMP     AL,31H              ;NO is byte 31H
        JZ      ONE BYTE            ;YES jump to send one byte
        CMP     AL,32H              ;NO is byte 32H
        JZ      ONE BYTE            ;YES jump to send one byte
        CMP     AL,33H              ;NO is byte 33H
        JZ      ONE BYTE            ;YES jump to send one byte
        CMP
```

```
                AL,34H              ;NO is byte 34H
        JZ      ONE BYTE            ;YES jump to send one byte
ERROR:  MOVE    AH,2
        MOV     AL,7                ;error routine to give one beep if wrong
        INT     21H                 ;code pressed
        CALL    WAIT1               ;wait one second
        JMP     W1                  ;jump to start again
ZERO BYTE:CALL SEND_BYTE            ;send type
        CALL    WAIT1               ;wait one second
        JMP     W1                  ;jump to start again
ONE BYTE:CALL SEND_BYTE             ;send type
W2:     MOV     AH,1
        INT     16H
        JZ      W2                  ;get next byte from kbd.
        MOV     AH,Ø
        INT     16H
        STOS    B
        MOV     CX,1
        JMP     PREP_PKT            ;jump to prepare packet
THREE BYTE:CALL SEND_BYTE           ;send type
        MOV     CX,5
W3:     MOV     AH,1
        INT     16H
        JZ      W3                  ;get five bytes from kbd.
        MOV     AH,Ø
        INT     16H
        STOS    B
        LOOP    W3
        MOV     CX,5
PREP_PKT:CALL SEND_OUT              ;send the packet out
        CMP     ERROR,1             ;is any error?
        JZ      PRCS_ERROR          ;YES , jump to process error
        CALL    WAIT1               ;wait one second
        JMP     W1                  ;jump to start again
PRCS_ERROR:MOV  DX,OFFSET ERROR_MSG      ;mov into DX offset of error message
        MOV     AH,9
        INT     21H                 ;dispaly error message
        CALL    WAIT1               ;wait one second
        JMP     W1                  ;jump to start again
GET_PKT:MOV     CX,MAX_RETRY        ;mov into CX the MAX retry count
TRY_AGAIN:PUSH  CX                  ;save count
        CALL    GET_IN              ;recieve the packet
        POP     CX                  ;get count
        CMP     NAK_ERROR,Ø         ;is nak_error flag set ?
        JZ      RECD_OK             ;NO, the pakcet is recd. O.K.
        LOOP    TRY_AGAIN           ;else loop to get packet 2 more times
        JMP     PR
```

```
                     JS_ERROR         ;max retry over, jump to proccess error
RECD_OK:CALL        _DISPLAY          ;call display routine
        CALL        WAIT1             ;wait one second
        JMP         W1                ;jump to loop again
CODE    ENDS
END     START
WAIT1:  PROC        NEAR
        XOR         CX,CX
        XOR         DX,DX
        MOV         BH,2DH
        INT         21H               ;set time to Ø
WAIT_CYCLE:MOV      AH,2CH            ;set AH for one sec delay
        INT         21
        CMP         DH,1              ;is delay over ?
        JL          WAIT_CYCLE        ;no, jump to wait cycle again
        RET                           ;YES return
WAIT15: PROC        NEAR
        XOR         CX,CX
        XOR         DX,DX
        MOV         AH,2DH
        INT         21H               ;set time to zero
WAIT11: MOV         AH,2CH            ;move count in AH to get 15 ms delay
        INT         21H
        CMP         DL,15             ;is 15ms over ?
        JG          WAIT_OVER         ;YES, jump to wait_over
        CALL        GET_BYTE
        CMP         AL,ACK            ;check if ACK recd.
        JZ          ACK_RECD          ;YES, jump to ACK recd.
        MOV         READ_ACK,Ø
        JMP         WAIT11            ;else jump to wait again
ACK_RECD:MOV        READ_ACK,1        ;set recd_ack flag to one
WAIT_OVER:RET                         ;return
        ENDP
```

APPENDIX-D

SOURCE - MODULAR MICRO-COMPUTE SYSTEM
TECHNICAL DESCRIPTION

[PROFESSIONAL ELECTRONIC PRODUCTS]

THE MODULAR MICRO COMPUTER SYSTEM TYPE EP- 131

A.1 INTRODUCTION

The NUCLEUS bus structure provides a common element for communication between a wide variety of system modules which includes: Processors, memory, digital input/ output, analog I/O, industrial I/O, peripheral controllers, plug-in power supply modules and misc. utility modules. Implementation of NUCLEUS system and nomenclature of different modules is shown in figure A-1.

The purpose of this application note is to help you develop a working knowledge of the NUCLEUS bus specification. This knowledge is essential for configuring a system containing multiple modules. This application note provides an in depth examination of the bus signals, operating characteristics, and bus interface circuits.

A.2 FEATURES

- Uses standard Euro size (100 x 220 mm and 233.4 x 220mm) cards
- Uses high reliability 2 part 64/ 96 pin standard Euro connector.
- Uses uSIC (Microprocessor based System for Industrial Control) compatible bus using well defined IEEE 796 (Intel Multibus) electrical signals on standard Euro size hardware. This bus is well defined for 8, 16 and also 32 bit systems allowing for easy upgradability of the hardware.

A.3 DESCRIPTION

The mechanical outline of NUCLEUS system single height shown in Fig. A-2. To support 8 bit hardware the signals on the row A and C remain used. The 16 bit modules use 96 pin connector (with rows A, B and C of 32 pins each) while 8 bit modules use 64 pin connector (with rows A and C). For bravity, the back panel connector pin assignments of 64 pin connector for 8 bit systems is reproduced in Table A-1.

The signals on the back panel are bussed i.e. pin 1 aof all the back panel connectors is connected together, similarly pin 2 of all connectors is connected together and so on. This rule has a few exceptions which have been mentioned with the description of the non- bussed signals. This bussing of the signals allows any module of the system to work in any slot on the sub-rack.

The NUCLEUS bus signal lines can be grouped in the following categories: address lines, bidirectional data lines, multilevel interrupt lines, and several bus control, timing and power supply lines. The address and data lines are driven by three- state devices, while the interrupt and some other control lines are open- collector driven.

The modules that use the NUCLEUS bus have a master- slave relationship. A bus master module can drive the command and address lines i.e. it can control the bus. A Processor module is an example of a bus master. A bus slave can not control the bus. Memory and I/O expansion boards are examples of bus slaves.

Notice that a system may have a number of bus masters. Bus arbitration results when more than one master request control of the bus at the same time. A bus clock is usually provided by one of the bus masters and may be derived independently from the processor clock. The bus clock provides a timing reference for resolving bus contention among multiple requests from bus masters. For example, a processor and a DMA (direct memory access) module may both request control of the bus. This feature allows different speed masters to share resources on the same bus. Actual transfers via the bus however, proceed asynchronously with respect to the bus clock. Thus, the transfer speed is dependent on the transmitting and receiving devices only. The bus design prevents slow master modules from being handicapped im their attempts to gain control of the bus, but doesnot restrict the speed at which faster modules can transfer data via the same bus. Once a bus request is granted, single or multiple read/ write transfers can proceed. The most obvious applications for the master- slave capabilities of the bus are multiprocessor configurations and high- speed direct- memory- access (DMA) operations. However, the master- slave capabilities of the bus are by no means limited to these two applications.

NUCLEUS bus for 8 bit systems

PIN NO.	ROW A	ROW B	ROW C
1	GND	-	GND
2	+5V	-	+5V
3	-15V	-	-15V
4	+15V	-	+15V
5	DT0/	-	DT1/
6	DT2/	-	DT3/
7	DT4/	-	DT5/
8	DT6/	-	DT7/
9	ADR0/	-	ADR1/
10	ADR2/	-	ADR3/
11	ADR4/	-	ADR5/
12	ADR6/	-	ADR7/
13	ADR8/	-	ADR9/
14	ADR10/	-	ADR11/
15	ADR12/	-	ADR13/
16	ADR14/	-	ADR15/
17	INT0/ OR NMI	-	INT1/
18	INT2/	-	INT3/
19	INT4/	-	INT5/
20	INT6/	-	INT7/
21	CCLK/	-	CBRQ/ OR HOLD/
22	INTA/ (*)	-	INH2/ (*)
23	XACK/ (*)	-	INH1/ (*)
24	IORC/	-	IOWC/
25	MRDC/	-	MWTC/
26	BUSY/ OR HLDA	-	BREQ/ (*)
27	BPRN/ (*)	-	BPRO/ (*)
28	BCLK/ (*)	-	INIT/
29	GND	-	GND
30	+5V	-	+5V
31	+5V	-	+5V
32	GND	-	GND

Note: (*) - Not used in LEVEL 1 modules

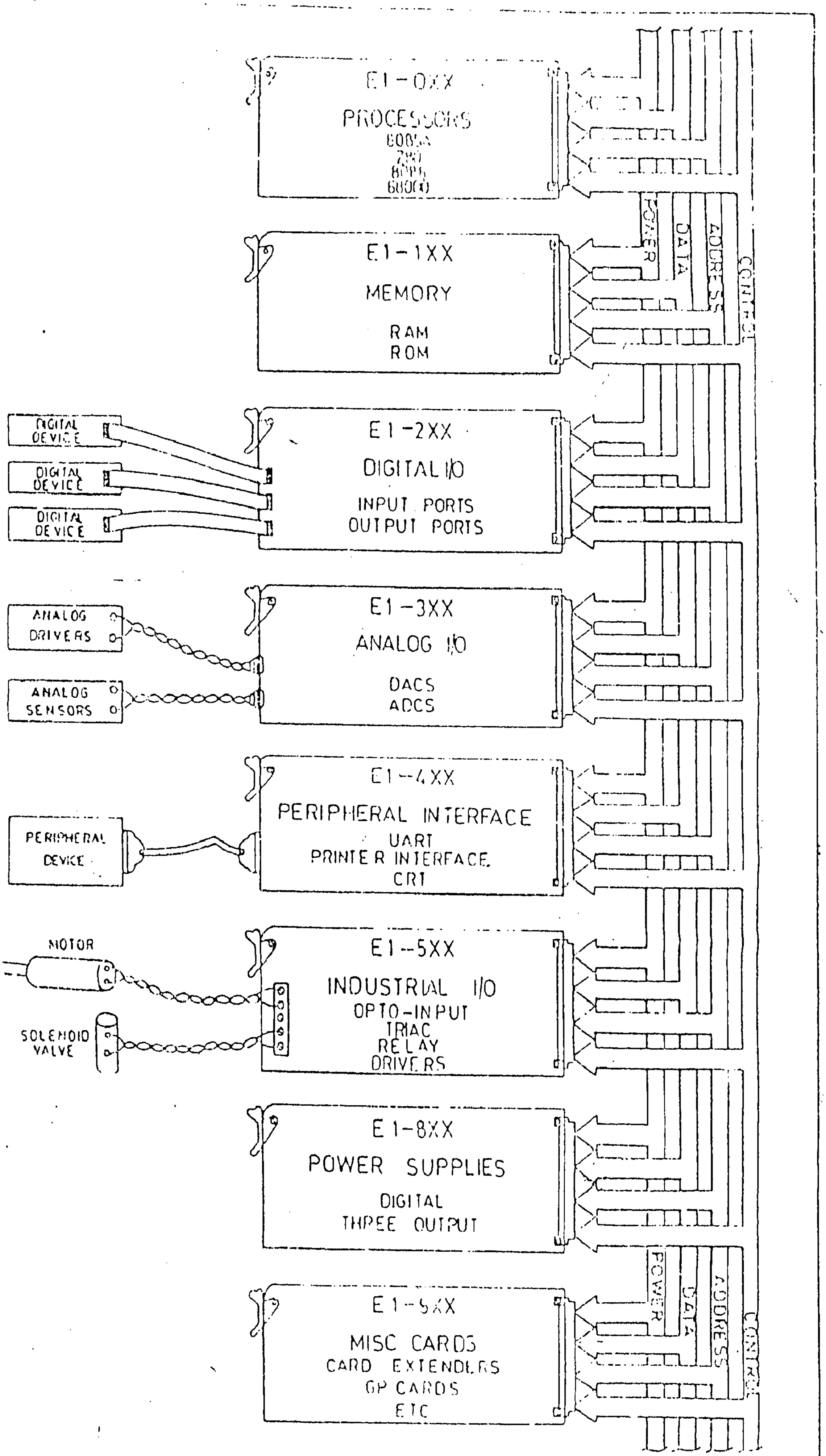
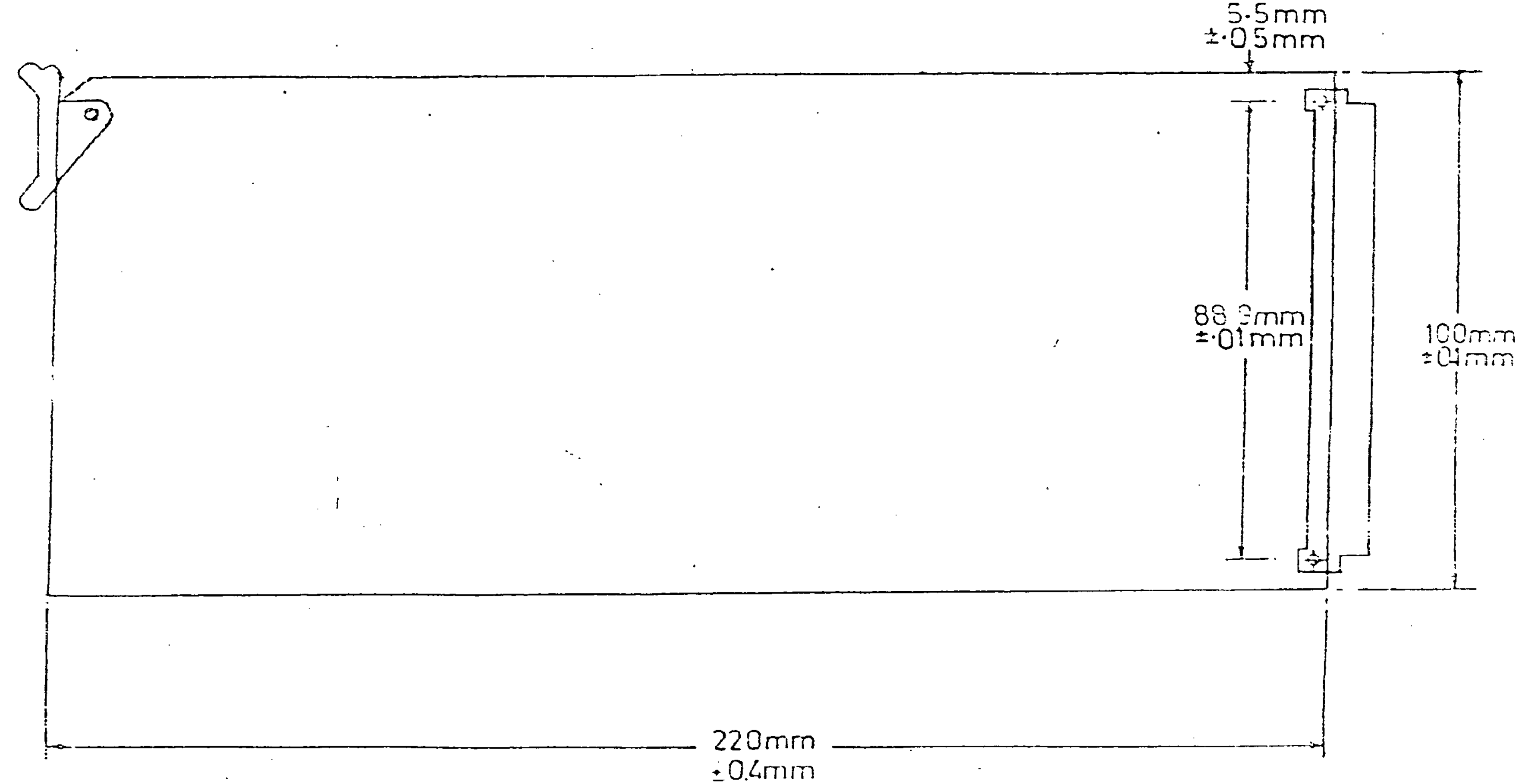


FIGURE-A-1 NUCLEUS BUS FOR 8-BIT SYSTEM



(LOOKING FROM COMPONENTS SIDE)

FIGURE-A-2 NUCLEUS BUS

* SINGLE HEIGHT CARD *

APPENDIX- E

SOURCE - PROGRMMERS REFERENCE MANUAL

FOR IBM PERSONAL COMPUTER

DOS Interrupt 21H
Invoke a DOS Function Request

Interrupt:

21H Invoke a DOS Function Request

DOS Version:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2

Description:

This interrupt is used to invoke all of the standard DOS function requests. Chapter 4 describes each of the function requests in detail and explains how to set up the registers properly so that you can invoke the function request you want.

Input:

Before issuing this interrupt, you must set registers as follows:

AH Place the number of the DOS function request in this register.

Set other registers as required by the individual function requests. Refer to Chapter 4 for details.

Output:

After control returns, the registers will be set as appropriate for each function request. Refer to Chapter 4 for details.

DOS Function Request 02H Display a Character

Function Request:

02H Display a character at the standard output device

DOS Version:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2

Description:

This function request displays a character at the standard output device (usually the display screen). It displays the character at the current location of the cursor and then advances the cursor one position.

If the character displayed is a backspace character, this function request moves the cursor one position to the left. However, it does not erase that character.

When the function request displays a character at the end of the line (the right edge of the screen), the cursor moves to the left edge of the next line. If the cursor is at the bottom right corner of the screen, the screen scrolls up one line when the cursor moves.

This function request checks for a Ctrl-Break after displaying the character. If the operator has entered a Ctrl-Break, the function request invokes an interrupt 23H to start the Ctrl-Break handler.

With DOS 2.0 (and later versions), you can redirect the standard output device. Redirection enables programs that use this function request to write to other devices, such as printers and disk files, instead of writing only to the screen.

Input:

Before invoking this function request, you must set registers as follows:

AH This register must contain 02H, indicating the number of this function request.

DL ASCII code for the character to be displayed.

DOS Function Request 09H Display a Character String at the Standard Output Device

Function Request:

09H Display a character string at the standard output device

DOS Version:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2

Description:

This function request displays a string of characters at the standard output device (usually the display screen). It displays the characters starting at the current location of the cursor and then advances the cursor one position.

To determine the end of the string, this function request assumes that the last character of the string is a $ character (ASCII code 24H). It does not display the $. Because of this string-ending convention, you should not use this function request to display information containing dollar signs, such as financial data.

If any character in the string is a backspace character, it causes the cursor to move one position to the left.

When the function request displays a character at the end of the line (the right edge of the screen), the cursor moves to the left edge of the next line. If the cursor is at the bottom right corner of the screen, the screen scrolls up one line when the cursor moves.

This function request checks for a Ctrl-Break after displaying the character. If the operator has entered a Ctrl-Break, the function request invokes an interrupt 23H to start the Ctrl-Break handler.

DOS 2.0 (and later versions) allows you to redirect the standard output device. Redirection enables programs that use this function request to write to other devices, such as printers and disk files, instead of writing only to the screen.

DOS Function Request 09H Display a Character String at the Standard Output Device

Input:

Before invoking this function request, you must set registers as follows:

AH This register must contain 09H, indicating the number of this function request.

DS:DX This register pair must point to the start of the character string in memory. The function request displays characters until it encounters a $ character (ASCII code 24H), which terminates the string. The $ character is not displayed.

Output:

None.

See Also:

02H—Display a character at the standard output device.

Example Programs:

Each of the following three examples uses function request 09H to display a string of characters terminated with a dollar sign on the screen. Most of the assembly language examples listed in this book use this function request to perform screen output.

Assembly Language Usage Example:

```
;
; DISPLAY STRING (09H)
;
code    segment public
assume cs:code,ds:code
        org     100h
start:  jmp     begin
msg     db      'Hi! This is a dollar sign terminated string.','$'
begin:  mov     ax,cs               ;set up ds
        mov     ds,ax
        mov     dx,offset msg       ;set up to display message
        mov     ah,09h              ;display string function request
        int     21h                 ;call DOS
```

DOS Function Request 2CH
Get the Time

Function Request:

2CH Get the time

DOS Version:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2

Description:

This function request returns the time (hours, minutes, seconds, and hundredths of seconds) as maintained by DOS. The DOS time is based on a value the user entered (or that was retrieved from a clock/calendar) after turning on or rebooting the computer.

Input:

Before invoking this function request, you must set the following:

AH This register must contain 2CH, indicating the number of this function request.

Output:

After control returns, the following are set:

CH Indicates the current hour in 24-hour format (0 through 23).

CL Indicates the current minute (0 through 59).

DH Indicates the current second (0 through 59).

DL Indicates the current 1/100 of a second (0 through 99).

See Also:

2AH—Get the date.
2BH—Set the date.
2DH—Set the time.

Example Programs:

Each of the following three examples uses function request 2CH to retrieve the current time. The program displays the time on the screen.

DOS Function Request 2DH
Set the Time

Function Request:

2DH Set the time

DOS Version:

1.0, 1.1, 2.0, 2.1, 3.0, 3.1, 3.2

Description:

This function request is similar to function request 2CH. It sets the DOS time.

This function request is useful for programs such as those that work with battery-powered clock/calendars. The program can retrieve the date and time from the clock/calendar and use this function request (and function request 2BH) to set the DOS date and time accordingly. If the program is invoked from the AUTOEXEC.BAT file, the user never needs to enter the date or time manually.

Programs can also use this function request to set the time to 0 in preparation for using the DOS clock as an event timer.

Input:

Before invoking this function request, you must set the following:

- AH This register must contain 2DH, indicating the number of this function request.
- CH This register must contain a value indicating the current hour in 24-hour format (0 through 23).
- CL This register must contain a value indicating the current minute (0 through 59).
- DH This register must contain a value indicating the current second (0 through 59).
- DL This register must contain a value indicating the current 1/100 of a second (0 through 99).

Output:

After control returns, the following is set:

AL Indicates the status of the operation, as follows:

00H The time you supplied was valid, and DOS was able to set its time accordingly.

FFH At least one of the time components you supplied was invalid (such as 25 hours or 63 seconds). The function request did not set the time.

See Also:

2AH—Get the date.
2BH—Set the date.
2CH—Get the time.

Example Programs:

Each of the following three examples uses function request 2DH to set the current time to 00:00:00.0.

Assembly Language Usage Example:

```
;
; SET TIME (2DH)
;
code    segment public
assume cs:code,ds:code
        org     100h
start:  jmp     begin
msg     db      'Time set to 00:00:00.00',0dh,0ah,'$'
begin:  mov     ax,cs              ;set up ds
        mov     ds,ax              ;to same as cs
        mov     ah,2dh             ;set time function request
        mov     ch,0               ;hours
        mov     cl,0               ;minutes
        mov     dh,0               ;seconds
        mov     dl,0               ;hundredths
        int     21h                ;call DOS
        mov     dx,offset msg      ;address of message
        mov     ah,09h             ;display string function request
```

BIOS Interrupt 14H
RS-232 Serial I/O

Interrupt:

14H RS-232 serial I/O

Description:

The BIOS function requests invoked via this interrupt let you control the actions of the RS-232 serial communications ports. These ports are actually standard communication paths that allow programs to communicate with modems, serial printers, and other computers.

There are four function requests associated with this interrupt. You choose the function request you want by setting the AH register to the appropriate value and then issuing an interrupt 14H. The following function requests are available via this interrupt:

AH	*Function Request*
00H	Initialize serial port.
01H	Send one character.
02H	Receive one character.
03H	Get serial port status.

The next several pages describe these RS-232 Serial Port function requests in detail.

BIOS Interrupt 14H
RS-232 Serial I/O
Function Request 00H—Initialize Serial Port

Interrupt:

14H RS-232 serial I/O

Function Request:

00H Initialize serial port

Computers:

PC, PCjr, XT, Portable, and AT

Description:

This function request sets up the serial port for transmission or reception of information. You can set the baud rate, parity, number of stop bits, and the character length.

Input:

Before invoking this function request, you must set the following:

AH This register must contain 00H to specify the Initialize Serial Port function request.

AL Set this register to contain the encoded initialization parameters, as follows:

Bits

7	6	5	4	3	2	1	0

Bit	*Description*
7–5	**Baud rate.** Set this three-bit field to indicate the number of characters that should be transmitted each second. Possible binary values include:

Value	*Baud Rate*
000	110
001	150

Value	*Baud Rate*
010	300
011	600
100	1200
101	2400
110	4800
111	9600

On the PCjr, only baud rates up to 4800 are supported. Attempts to set the baud rate to a higher value actually set the baud rate to 4800.

4–3 **Parity.** Set this two-bit field to indicate the parity checking scheme used by the serial port to ensure that no data is lost during transmission. Possible binary values include:

Value	*Parity*
00	No parity checking.
01	Odd parity.
10	No parity checking.
11	Even parity.

2 **Number of stop bits.** Set this bit to specify the number of bits that are sent after each character to indicate the end of that character. Possible values are:

Value	*Number of stop bits*
0	One stop bit is used.
1	Two stop bits are used.

1–0 **Character length.** Because information is sent over a serial line one bit at a time, the processor must know how many bits are contained in each character. Set this two-bit field to one of the following binary values:

BIOS Interrupt 14H
RS-232 Serial I/O
Function Request 00H—Initialize Serial Port

Value	*Character size*
10	seven-bit characters (standard ASCII)
11	eight-bit characters

DX Set this register to indicate the serial port you want to initialize, as follows:

0	The first (or only) serial port.
1	The second serial port.

Output:

The serial communications port is set up as directed. In addition, the following registers contain status information. This information is the same as that returned by function request 03H (Get Serial Port Status).

AH This register contains the encoded line status, as follows. The status information is true if the corresponding bit is set.

Bits

7	6	5	4	3	2	1	0

Bit	*Description*
7	Time-out error has occurred.
6	Transfer shift register is empty.
5	Transfer holding register empty.
4	Break occurred.
3	Framing error occurred.
2	Parity error occurred.
1	Overrun error occurred.
0	Data is ready.

AL This register contains the encoded modem status, as follows. The status information is true if the corresponding bit is set.

Bits

7	6	5	4	3	2	1	0

Bit	*Description*
7	Received line signal detect (a carrier signal was detected).
6	Ring indicator was detected.
5	Data set is ready (DSR).
4	Clear to send (CTS).
3	A change in the receive line signal detect (carrier signal) occurred.
2	Trailing edge ring detector.
1	A change in the data set ready (DSR) signal occurred.
0	A change in the clear to send (CTS) signal occurred.

See Also:

Interrupt 14H, function request 03H—Get serial port status.

BIOS Interrupt 14H
RS-232 Serial I/O
Function Request 01H—Send One Character

Interrupt:

14H RS-232 serial I/O

Function Request:

01H Send one character

Computers:

PC, PCjr, XT, Portable, and AT

Description:

This function request sends a single character through the RS-232 serial port.

Input:

Before invoking this function request, you must set the following:

AH This register must contain 01H to specify the Send One Character function request.

AL Set this register to contain the ASCII code for the character you wish to send.

DX Set this register to indicate the serial port to which you want to send the character, as follows:

0 The first (or only) serial port.

1 The second serial port.

Output:

Unless an error occurs, the character is sent to the serial port.

Error Conditions:

The following register indicates the success or failure of this function request:

AH If this register is set to 0, no error has occurred. However, if bit 7 is set to 1, an error of some sort has occurred. The remaining bits are encoded to indicate the line status, as follows:

Bits

7	6	5	4	3	2	1	0

Bit	*Description*
6	Transfer shift register empty.
5	Transfer holding register empty.
4	Break-detect error.
3	Framing error.
2	Parity error.
1	Overrun error.
0	Data is ready.

The errors reported in this register are a subset of the ones reported by function request 03H (Get Serial Port Status). The one error condition that cannot be reported here is the time-out error, which is normally reported in bit 7. Function request 01H uses bit 7 as a general error flag and therefore cannot use it to report a specific error. To ensure a complete error report, check only bit 7 of this register. If bit 7 is set, invoke function request 03H to get the complete error status.

See Also:

Interrupt 14H, function request 03H—Get serial port status.

BIOS Interrupt 14H
RS-232 Serial I/O
Function Request 02H—Receive One Character

Interrupt:

14H RS-232 serial I/O

Function Request:

02H Receive one character

Computers:

PC, PCjr, XT, Portable, and AT

Description:

This function request receives one character from the serial port. When the function request gains control, it waits until a character is available from the serial port, or until a time-out occurs. Therefore, if you don't want to be forced to wait, you should invoke function request 03H (Get Serial Port Status) first to determine whether data is ready to be received.

Input:

Before invoking this function request, you must set the following:

AH This register must contain 02H to specify the Receive One Character function request.

DX Set this register to indicate the serial port from which you want to receive the character, as follows:

0 The first (or only) serial port.

1 The second serial port.

Output:

Unless an error occurs, the following register is set after control returns:

AL This register contains the ASCII code for the character that was received.

Error Conditions:

The following register indicates the success or failure of this function request:

AH If this register is set to 0, no error has occurred. However, if bit 7 is set to 1, an error of some sort has occurred. The remaining bits are encoded to indicate the line status, as follows:

Bits

7	6	5	4	3	2	1	0

Bit	*Description*
6	Transfer shift register empty.
5	Transfer holding register empty.
4	Break-detect error.
3	Framing error.
2	Parity error.
1	Overrun error.
0	Data is ready.

The errors reported in this register are a subset of the ones reported by function request 03H (Get Serial Port Status). The one error condition that cannot be reported here is the time-out error, which is normally reported in bit 7. Function request 01H uses bit 7 as a general error flag and therefore cannot use it to report a specific error. To ensure a complete error report, check only bit 7 of this register. If bit 7 is set, invoke function request 03H to get the complete error status.

See Also:

Interrupt 14H, function request 03H—Get serial port status.

BIOS Interrupt 16H
Keyboard I/O

Interrupt:

16H Keyboard I/O

Description:

The BIOS function requests invoked via this interrupt allow you to receive characters from the keyboard and check to see whether a keyboard entry has been made. With the BIOS function requests, as opposed to the DOS function requests that perform the same operations, you can skip some of the function processing that DOS performs automatically. For example, if a Ctrl-C is pressed, the DOS character-handling routines automatically assume that character to be a special signal; that is, to abort the program. However, the BIOS function requests make no assumptions about the characters they receive.

You choose the function request you want by setting the AH register to the appropriate value and issuing an interrupt 16H. The function requests available via this interrupt are the following:

AH	*Function Request*
00H	Read next keyboard character.
01H	Determine whether character is available.
02H	Get current shift status.

The next several pages describe these keyboard I/O function requests in detail.

BIOS Interrupt 16H
Keyboard I/O
Function Request 00H—Read Next Keyboard Character

Input:

Before invoking this function request, you must set the following:

AH This register must contain 00H to specify the Read Next Keyboard function request.

Output:

Control does not return until a character is typed at the keyboard. After control returns, the following registers are set:

AL This register contains the ASCII code of the key that was pressed. If the key does not correspond to one of the 256 ASCII characters, this register will be set to 0.

AH This register contains the extended code for the character. This will either be the keyboard scan code shown in Figure 5–1 or 5–2, or an extended code listed in Table 5–2.

See Also:

Interrupt 16H, function request 01H—Determine whether character is available.
Interrupt 16H, function request 02H—Get current shift status.

BIOS Interrupt 16H Keyboard I/O Function Request 00H—Read Next Keyboard Character

Interrupt:

16H Keyboard I/O

Function Request:

00H Read next keyboard character

Computers:

PC, PCjr, XT, Portable, and AT

Description:

This function request reads a character typed at the keyboard. If the character has already been typed, and resides in the BIOS keyboard buffer, the character is returned immediately. Otherwise, this function request waits until a character is typed.

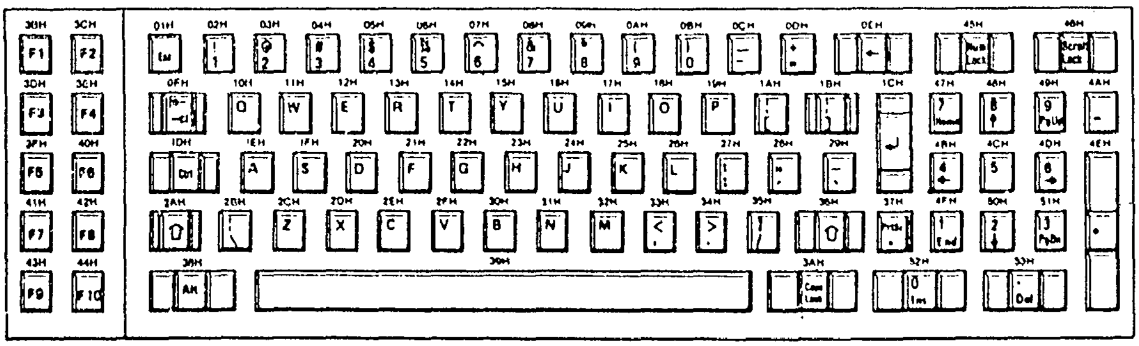
This function request returns two pieces of information about the character typed: its ASCII code and its extended code. The ASCII code is the standard code by which the character is known in many programming languages and computer systems. The extended code is the specific code that the BIOS uses to refer to an individual pressed key or key combination.

Both the ASCII and extended codes are returned for a couple of reasons. First, not all the keys on the IBM keyboard correspond to ASCII characters. For those characters and for certain combinations of characters and the Shift, Ctrl, or Alt key, the ASCII code is returned as a 0, and only the extended code serves to distinguish the character. Second, some of the keys, such as the numeric keys, the shift keys, and the asterisk keys, are duplicated on the IBM keyboard. Their ASCII codes are the same, but their extended codes are different. By returning both the ASCII code and the extended code, you can tell exactly which key has been pressed. For example, you can differentiate between the numbers on the numeric keypad and the numbers on the top row.

In most instances, the extended code returned by this function request is the keyboard scan code of the primary key that was pressed. The keyboard scan codes are specific codes that the IBM keyboard sends in response to a pressed key. Keyboard scan codes apply only to the individual keys. There are no separate codes for uppercase (shifted) characters or characters entered while the Ctrl or Alt key is pressed. In most cases, to determine whether the Shift, Ctrl, or Alt key was pressed with another key, you must use function request 02H (Get Shift Status).

Figures 5–1 and 5–2 show the keyboard scan codes for the two standard IBM keyboards. Figure 5–1 illustrates the keyboard used for the IBM PC, XT, and Portable. Figure 5–2 illustrates the AT keyboard.

Figure 5–1. Keyboard Scan Codes for the IBM PC, XT, and Portable



For some key combinations, where both a standard key and a shift key (either the Shift, Alt, or Ctrl key) are pressed at the same time, this function request returns an extended code that is different from the standard keyboard scan code. Table 5–2 lists those key combinations and extended codes.

If your program is not willing to wait until a key is pressed, it can invoke function request 01H (Determine Whether Character is Available) to determine whether a character is waiting to be received.

Figure 5–2. Keyboard Scan Codes for the IBM AT

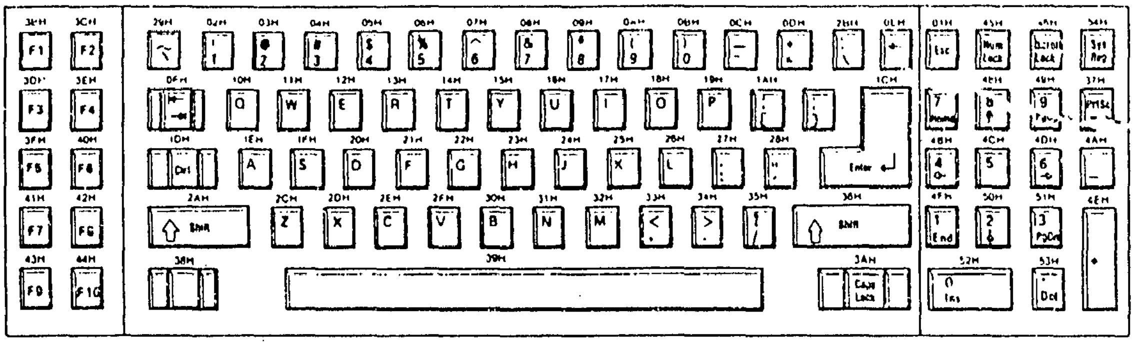


Table 5–2 Extended Codes for Special Key Combinations

Extended Code	*Character Combination*
03H	Nul character.
54H–5DH	Shift plus F1 through F10.
5EH–67H	Ctrl plus F1 through F10.
68H–71H	Alt plus F1 through F10.
72H	Ctrl plus PrtSc.
73H	Ctrl plus left arrow.
74H	Ctrl plus right arrow.
75H	Ctrl plus End.
76H	Ctrl plus PgDn.
77H	Ctrl plus Home.
78H–83H	Alt plus 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -, =
84H	Ctrl plus PgUp.

APPENDIX - F

SOURCE - GRAPH X MANUAL

Main Features of GRAPH-X

Graph-X can be used for plotting points drawing lines, circles, arcs and rectangles. It allows text writing. It allow to switch colour levels from dark to green.

On Screen graphic facilities

Following on screen graphic facilities are available:

INT 10.COM	-	extension to DOS for on screen graphics
FOR 2 GRH, OBJ	-	linkage to FORTRAN
HCG. COM	-	confugration file for graphiic card.

Printing facilities

Following print facilities are available :

HARCOPY. COM	-	extension to DOS for screen printing
PRINTER. DEF	-	definition file for IBM or Epson dot matrix printer.

Both INTIO.COM and HARCOPY. COM must be loaded before using graphics.

Coordinate system:

Graphic screen may be considered as two dimension array with X being horizontal and Y-being vertical axis. The origin of the axis is the upper left corner. The addressable pixels are:

X(horizontal) : 0 to 719

Y(vertical) : 0 to 347

Any part of figure outside the legal boundary is clipped at the edges.

Any letter crossing the boundary is suppressed fully.

Height of the pixel is 1.5 times more than to its width. To draw figures with same scales on both axises, vertical dimensions are multiplied 2/3.

Screen Colour Level

Figures may be drawn green or black in any background, also the figure may be drawn in colour opposite to the background. The colour levels are :

0	-	causes the figure to be drawn black
1	-	causes the figure to be drawn green
2	-	causes pixel to change colour it causes green figures on black colour and black figures on green background

Repeated drawing of a figure with level 2 will cause it to blink.

Text Writing:

Text is drawn 9 pixel wide and 14 pixel high. Background and the colour of the text depends on the set level. The character must be inputed in ASCII form. If the character are more they may be defined as double precision or character.

Graphic pages:

There are two independent pages in which graphic display is stored. Each graphic page is independent and requires 16 K memory. Both these pages can be used independently for display and writing. Caution has to be taken while using page 0. The first 4K of page 0 is shared by text mode buffer. Any text mode operation will contaminate graphic page 0. The run time error are eclipsed when the screen is in graphic mode.

Using GRAPH-X with fortran:

Few precaution are necessary for using GRAPH-X with FORTRAN.

Each function or subroutine of GRAPH-X must be declared as EXTERNAL in the program. Variables should not be passed directly to GRAPH-X routine. Number must be assigned to a variable, which is then passed as an argument to the routine.

GRAPH-X defined subroutine:

GRAPH-X provide subroutine to set up graphics, draw figures and print hard copy. The subroutines defined by GRAPH-X are described below :

Subroutine ARC:

This subroutine draws a quarter circle on the screen give the centre of the arc radius and the guardrant in which the arc is to be drawn:

Calling sequence: CALL ARC(X,Y,RADIUS,QUAD)

Arguments:

X,Y	=	Coordinates of Centre of the arc.
RADIUS	=	Number of pixel between point X,Y and the arc in X-ax is
QUAD	=	Number of quadrant in which arc is to be drawn. Numbers are from 1 through 4 with first quadrant as 1 and increasing counter clockwise.

Subroutine BLKFIL

Given height, width and the lower left corner, this subroutine draws a solid rectangle.

Calling sequence: CALL BLKFIL(X,Y, WIDTH, HEIGHT)

Arguments :

X,Y	=	Coordinates of lower left corner
WIDTH	=	width in pixels
HEIGHT	=	height in pixels.

Subroutine CIRC

Given the centre and the radius the subroutine draws a circle

Calling sequence: CALL CIRC(X,Y, RADIUS)

X,Y	=	coordinates of centre point
RADIUS	=	number of pixels between centre and circumference on X-axis

Subroutine CLSCR

This subroutine clears the graphic page currently written into :

Calling sequence: CALL CLRSCR

Subroutine DISP

This subroutine set the page to be displayed on the screen

Calling sequence : CALL DISP(BUFPAGE)

Arguments :

BUFPAGE = 0 or 1 buffer page

Subroutine DLINE:

This subroutine draws a line from current cursor position to specified cursor position.

Calling sequence: CALL DLINE (X,Y)

Arguments : X,Y = coordinates of end position

Subroutine FILL:

This subroutine fills the area in a convexpolygon by reverse colour, given a point inside the polygon.

Calling sequence: CALL FILL(X,Y)

Arguments:

X,Y = Coordinates a of a point lying inside the corvex polygon.

Subroutine PUTPUT arguments;

This subroutine moves the imaginary cursor to specified position on the screen may be used with DLINE subroutine.

Calling sequence: CALL PUTPUT(X,Y)

Arguments:

X,Y = Coordinates of the imaginary cursor

Subroutine GETPT:

This subroutine reads the intensity of a given pixel any where within legal coordinates.

Calling sequence : CALL GETPT(X,Y) INTENS)

Arguments:

X,Y = Coordinate of the point whose intensity is needed

INTENS = Intensity value

0 - Black

1 - Green

Subroutine GMODE

This subroutine puts graphic card into graphic mode. It must be called before calling any graphic function or subroutine.

Calling sequence :

CALL GMODE

Subroutine GPAGE

Subroutine determine the page to be written into

Calling sequence : CALL GPAGE (BUFPAGE)

Arguments:

BUFPAGE = Buffer page 0 or 1

Subroutine LEVEL

Subroutines sets the level to be used by subsequent graphic function.

Calling sequence : CALL LEVEL (INTENS)

Arguments:

INTENS = 0 black

1 Green

2 XOR

Subroutine PLOT

Plots a given point on the screen depending on the colour level set.

Calling sequence : CALL PLOT (X,Y)

Arguments:

X,Y = Coordinates of points to be plotted.

Subroutine TMODE;

This subroutine puts graphic card into normal text mode:

Calling sequence : CALL TMODE

Subroutine TEXTF:

This subroutine writes array of characters at a desired point on screen character are written horizontally. Upto 4 character may be accommodated in single precision, 8 characters in double precision and upto 132 in CHARACTER xn declaration.

Calling sequence : CALL TEXTP(X,Y,LEN, MSG)

Arguments :

X,Y	=	Coordinate of first letter in the array. The coordinate is of lower left corner.
LEN	=	Length of character in text string
MSG	=	Array of character

Subroutine HRDCPY

This subroutine generates hard copy of the graphic page on the dot matrix printer.

Calling sequence : CALL HRDCPY (OPT CHAR)

Arguments :

OPTCHAR	=	'1' Graphic page 0
		'2' Graphic page 1
		'3' Graphic page 0 reverse video
		'4' Graphic page 1 reverse video
		'5' graphic page 1 and page 0
		'6' Graphic page 0 and page 8 reverse video.