5-31-1991

Interfacing plasticating extruders with personal computers for data acquisition, information analysis and retrieval purposes

Gautamkumar Y. Shah
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Interfacing Plasticating Extruders with Personal Computers for Data Acquisition, Information Analysis and Retrieval Purposes.

Gautamkumar Y. Shah, M.S. Manufacturing Engineering Systems.(1991)

Thesis Advisor: Dr.Keith O'Brien.
Professor of
Mechanical Engineering

The current manufacturing industries are influenced more and more by computers and automation. Two of the major areas of computer uses are data acquisition and interfacing, both lead towards automation.

The intent of this thesis is to focus on the area of interfacing personal computers with plasticating extruders for information retrieval and storage. The requirements for software and hardware are discussed. Analysis of the stored information can be used to improve the quality of the final product. Implications of the stored parameters on the final product are also discussed. A sample program to store and retrieve the important parameters is demonstrated.

INTERFACING PLASTICATING EXTRUDERS WITH PERSONAL COMPUTERS FOR DATA ACQUISITION,INFORMATION ANALYSIS AND RETRIEVAL PURPOSES

by

Gautamkumar Y. Shah.

Thesis submitted to the faculty of the Graduate School of
the New Jersey Institute of Technology in partial fulfillment of
the requirements for the degree of
Master of Science in Manufacturing Engineering Systems.
May 1991

Approval Sheet

Title of Thesis: Including Plasticity Effects in [illegible] Personal Computer For Data Acquisition, [illegible]

Name of Candidate: [illegible]

Thesis and Abstract Approved:

Dr. Keith O'Brien [illegible] Date

Professor of Mechanical Engineering

Dr. Harmeet Lee [illegible] Date

Associate Professor of Mechanical Engineering

Dr. R. Chen Date

Professor of Mechanical Engineering

VITA

Name: Gautamkumar Y. Shah.

Permanent address:

Degree and date conferred: Master of Science, May 1991.

Date of birth:

Place of birth:

Secondary education: R.P.T.P. Science College,
Vallabh Vidya Nagar, India.

Collegiate institution attended	Date	Degree	Date of degree
New Jersey Institute of Technology	09/89 to 05/91	M.S.	May 1991
Bangalore University	08/84 to 12/88	B.S.	Jan 1989

Major: M.S. Manufacturing Engineering.
B.S. Electronics Engineering.

ACKNOWLEDGEMENTS

I would like to thank Dr. Keith O'Brien, for his immense help and cooperation. I gratefully acknowledge his help for suggesting the topic and providing the necessary guidance for the successful completion of this program.

I would like to express appreciation to Dr. Nouri Levy and Dr. R. Chen of my thesis committee for their valuable suggestions. I also wish to thank Mr. Charlie Martin and staff members of Killion Extruders Inc, for providing valuable information and suggestions.

I am greatly indebted to my family their moral and monetary support. Last but not the least I would like to thank all my friends who helped me one way or other during the course of this program.

Contents

List of Figures

Chapter 1

INTRODUCTION

Increasingly, computers are touching more and more aspects of our private and professional lives. That they have revolutionized our documents, library, and information storage retrieval techniques is evident. That they have affected profoundly the very nature of professions transfer, is a prospect which this professions cannot afford to ignore. Information handling mainly deals with the storing and retrieving of the information or the given data. Information storage refers to the mechanism of storing the data in the computer memory. Information retrieval is the name for process or method by which a prospective user of information is able to convert his need for his information into his actual list of citation from the items in the storage. It is the finding or discovery process with respect to stored information and can be considered another, more general name for production of a demand bibliography. Information retrieval, however arrives at a general class of information by examining specific units of input data for their interrelationships. There are number of methods of retrieving the information implementation of which depends upon the user requirement. That the customer is always right is one of the perti-

nent of the business. Accordingly more and more industries are building database to assure that they are doing right to their customers. These database can contain the information on who the customer is, who the key contacts are, what the customer is buying and how often, when the last purchase was made, amount of the average order, source of the customer inquiry, and possibility of hundreds of other demographic items. Over the past decades technology advances have given personal computers, enormous data gathering and storage capability, making data -collection, storage and analysis one of the PC's most prominent task.

1.1 Information Analysis

In order to perform the satisfactory analysis of the stored information, it is generally necessary to identify the basic elements which are used to represent the information and to recognize the rules by which the basic elements can be combined into large units. The development of the effective scheme must take into account the multidimensional nature of the documented information. Merely establishing a multidimensional system for analyzing information is not enough. It is necessary that information be analyzed so that any question which may be posed is clearly and obviously in harmony with the system of analysis or at least capable of being brought into harmony with it. The essence of an information storage and retrieval system is its database or collection of files. If an information system is to provide timely and accurate information to all levels of users, the database must yield the desired data quickly and efficiently, also the database management system must be flexible enough to allow growth and changes. Database management is thus concerned with the problems of storing data in farm as flexible and accessible as possible.

1.2 Industrial Scenario

Killion Extruders Inc. is a company which manufactures extrusion equipment according to the customer specified dimensions and supplies them all over the world. All the equipments are made to orders. Due to unique modularity for all the customers it is needless to say that specifications are different for all the customers. For example specified screw diameter can be different for different customers, also L/D dimensions may vary according to the need of the customer. As we can see number of different variables are encountered in the design of a particular equipment and these values keep on increasing as number of customers increases. It becomes enormously difficult to keep track of all the variables and seek any one or more of them whenever required quickly.

For such an industry, it is required to have an excellent information retrieval system which can extract any information for any customer or an equipment. With a good information retrieval system It becomes much more easier for a design engineer or a marketing executive or any employee to retrieve an information for any customer or any equipment design. In other words an efficient database which can access the required data for the customer or the necessary information for the equipment can be of much more help to increase the speed of communication as well as it can lead towards faster designing process. Here by the term faster designing process I mean that if, a part is to be used in certain product and if a design engineer wants to know in previous cases where the particular part was used, by just pressing the part he or she can get all the information about the same. Finally we can say that the data base should be designed in such a manner that it reduces the time spent defining and implementing an information processing system; to use the normal problem solving approach to obtain better information; and to increase

the control over the creation and maintenance of data file and over access to data and procedures for processing data.

1.3 Data Acquisition

Data acquisition is a process of accessing required data and storing them in the computer. There are many methods of extracting data. One which can be implemented for our purpose is called on line data acquisition. Here required data is acquired and stored when the process is running. If we see this in context of plastic extruders, it is important for us to control access the data online. Storing these parameters can help us analyzing the process which can lead us towards better quality control and other benefits. As mentioned earlier, important parameters are displayed on the control penal. In order to store them in required fashion we need to do is

1.3.1 Hardware Design

By which we can interface data displayed on the control panel to the computer.

1.3.2 Software Design

After proper interfacing of computer and control panel we need a software which enables us to read the data and converts it in the form of information we need. Also the software should store the data hence it can be retrieved back when needed.

Chapter 2

EXTRUSION PROCESS

2.1 Extrusion Process

The word extrude originates in the Latin words, 'ex' (out) and 'trudere' (to thrust). This explains the process itself as shaping by forcing through a die. The screw extruder efficiently and continuously converts the solid polymers in to melt and pumps the very high viscosity melt through a die at high pressure. A typical plastic extruder contains following parts 1) Hopper 2) Barrel and screw 3) Control panel 4) Motor

2.1.1 Hopper

It is a cylindrical funnel type vessel, which is mounted on the top of barrel and the raw material or polymers are funneled through it. They are passed through the barrel in which the screw rotates and conveys it towards the die.

2.1.2 Barrel and Screw

It is the most important part of the extruder machine. Extruder machine ranges from 1" to 12" in diameter. It could be build for higher ranges. The solid polymers fed from the hopper is melted, homogenized, and pumped through the die at high pressure and temperature through this section. Here the extruder is more than just a pump, it also supplies energy to melt. There are two sources of energy 1) Hot barrel of the extruder which is equipped with heaters and thermal energy and 2) Mechanical energy is introduced through rotating shaft. The solids and melt are pumped forward by relative motion of the screw and barrel. The screw 'floats' in the barrel with a small clearance and material act as a lubricant. Screw and barrel are made up of steel. The frequency of screw rotation at which the extruder will operate depends on the size of the extruder required production rate. The melted material which is also called melt is conveyed to the die and the shape is given. The process could be analyzed in three parts solid conveying, melting, and melt conveying.

In solid conveying the solid material or polymers are fed to the screw and barrel through hopper. The process of converting solid material to melt is called melting and the process of conveying molten polymers to the die is called melt conveying.

This was the basic idea about how the extrusion process works. The major part of the thesis deals with the storing important parameters during the process. (On line) The important parameters are

- Barrel Temperature in Zone 1
- Barrel Temperature in Zone 2

- Barrel Temperature in Zone 3
- Barrel Temperature in Zone 4
- Temperature at Die.
- Temperature of melt.
- Screw Rotation or (RPM)
- Ampere.
- Pressure developed in the barrel

2.1.3 Control Panel

Third and equally important part is control panel. All the parameters mentioned above are displayed on the control panel. The display will be either in the digital or analog form. Control panel allows us to control the parameters.

2.2 Screw Design

The screw could be designed of various types. It is a question of debate whether the screw design should be able to handle a range of materials, or it should be dedicated to a single product.

In general terms, blown film screws must provide excellent homogeneity and cooler melt temperature perhaps, with multiple mixing section for the greatest possible melt strength. Coating and laminating screws will have shallower channel depths for applying hot melts at over 600 F. Sheet and profile screws fall somewhere in between.

2.2.1 Low-shear mixing sections

Several machines and component builders have introduced dynamic mixing sections to promote homogeneity while keeping melt temperature low. In this method indentations are provided in the screw and/or barrel, either at the discharge of the extruder or just prior to vent for devolatilization. This method helps for blown film where low processing temperature and good homogeneity are required for melt strength in bubble. This dynamic mixers are particularly helpful in processing high molecular weight where temperature should be kept below 450 F.

2.2.2 Single screws for different polymers

An innovative tool varies the height of flow restrictors at the end of the extruder, to optimize the pressure inside the barrel. Made by R.Dray Mfg.[1][13], it has a hydraulically operated shaft that runs through the barrel to move the retractors. Adjusting the pressure in this way lets processor use one screw and barrel on material with different viscosities, temperatures, throughputs and different requirements.

Some companies have established direct links between simulation/design software and metal cutting mills at screw fabricating plant.[14], which means processors can now design and build screw in one step. Simulation software enables processors to optimize the screw with their specific process condition.

2.3 Theoretical Aspects

The forward motion of the material along the screw of an extruder may likened to the motion of a nut on a screw in a hexagonal tube. As the screw turns the nut advances in the constraining tube.

[1]Refernces at the end of the text

In the cylinder of the extruder the material must slip along the screw and be kept from too great lateral turning by the friction against the side of the cylinder. This means that the total friction between screw and material should be less than that of material and cylinder wall, the surface area of each screw flight should be less than the cylinder wall surface area facing it. This relation between areas forms one of the basic considerations in screw and cylinder design. The wider the space between the flights and the screw, the lower will be the ratio of the flights surface area to subtended cylinder surface area.

The coefficient of friction often is not the same for material against screw and the material against the cylinder. This is not only because of the difference in material but also due to difference in temperature. Some materials have a very heat sensitive coefficient of friction. The relation between p,the pressure in the die chamber; D, the screw diameter; μ, the viscosity of the material; and q, the rate of extrusion in cubic units may be expressed in the form of a dimensionally homogeneous power series of type

$$\mathrm{p} = \sum(Aq^y D^x N\omega)$$

where N is the speed of the rotation of the screw and A is dimensionless constant. The following are the dimensions of the units involved:

$p = \frac{m}{LT^2}$;

$\mu = \frac{m}{LT}$;

$D = L$;

$N = \frac{1}{T}$;

$q = \frac{L^3}{T}$;

Viscosity is defined as the "the ratio of shear stress to rate of shear", and a plastic material as " one which shows no appreciable flow below a certain finite stress". A

viscous fluid will be understood to show appreciable flow at all stresses. It should be noted that no account has been taken of density it can usually be omitted in such consideration when the speed at which material advances slow viscosity is high. Since the power series must be dimensionally identical with p, it follows that

$$\mathrm{p} = \mu N \phi \left(\frac{q}{ND^3}\right)$$

The function ϕ may be determined by consideration of the limiting cases. With die fully closed, $p = \mu N \Phi(0)$, so that, if p is to have a finite value, $\Phi(0) = k$ or $p \propto \mu N$, a fact that does not appear to be generally realized. When die is fully or partly opened, $\frac{p}{\mu N}$s a constant, all other conditions remaining steady. Hence Φ is a constant, so that necessarily, $q \propto ND^3$. If this analysis is carried further and applied to the movement of plastic between the die chamber and die, the pressure can be expressed as the function of D, μ, and q only. The relation is found to be $p = K\frac{q\mu}{d^3}$

It must be stressed that these conclusions are drawn for an assumption of geometrical similarity[4]. Consequently, any practical attempt to study it with the use of one machine only must be carried out by varying N and μ. Variation q by means of bleed-off cock will destroy the condition of geometrical similarity.

2.4 Effect of Temperature on Flow Properties

Temperature is one of the most critical operating variables in any processing method. This is the consequence of the dependence of the flow properties on temperature. Viscosity on polymer melts, like that if liquid generally decreases with temperature. Eyring and co-workers [2] developed an approximate theory for liquids which permits a rough estimation of the viscosity on temperature[3]. It also predicts an Arrhenius type dependence of viscosity on temperature:

[2]References listed at the end of the text.

$$\mu = Ae^{\frac{E}{RT}}$$

Where R is the gas constant, A is coefficient dependent on the fluid, and E is an activation energy. If the fluid is nonNewtonian [3], the viscosity , μ in the above eqn. can be replaced by the apparent viscosity η . In such a case two equations result, because the activation energy E assumes different values if temperature is varied with constant shear rate or constant shear stress :

$\eta = Ae^{\frac{E_\tau}{RT}}$

and

$\eta = Ae^{\frac{E_\gamma}{RT}}$

For fluids following above mentioned equ., the ratio of activation energy is

$\frac{E_\tau}{E_\gamma} = 1 - \gamma \frac{\delta_\eta}{\delta_\tau T}$

For a pseudoplastic fluid , $\frac{\delta_\eta}{\delta_\tau T} < 0$, and therefore $E_\tau > E_\gamma$. If the fluid behaves according to the power low model, the following relationship exists between the activation energies:

$E_\tau = \frac{1}{n} E_\gamma$

The activation energy of a nonNewtonian fluid varies with temperature in addition to shear rate or shear stress. An increase in temperature causes a decrease in activation energy and also tends to suppress nonNewtonian effects. Therefore if wide temperature ranges are encount ered, the activation energy and the fluid models parameters have to be repeatedly reevaluated under local conditions in relatively narrow temperature ranges.

Hopkins [3] showed that the temperature dependence of apparent viscosity is incorporated in flow equations, it often leads to mathematical difficulties. A more convenient empirical form of temperature dependence is therefore, frequently used:

[3]The term nonNewtonian refers to all those fluids which , for various reasons, do not follow Newton's law of viscosity.

$\eta = \eta_0 e^{a^\star(T-T_0)}$ Where η_0 is the apparent viscosity at $T = T_0$. The constant $a^\star$ has a negative value. By expressing temperature dependence in this way ,$a^\star$, will also assume different values at constant shear rate and shear stress.

2.5 Effect of Pressure on Flow Properties

The viscosity of liquids depends on the intermolecular forces, which in turn are dependent on intermolecular distances. It is not surprising, therefore to find, that the compression of liquids tend to increase their viscosity. Moreover, the compressibility of polymer melts at processing temperature is higher than that of ordinary liquids. Hence a greater effect of pressure on viscosity can be expected. Eyrings' "hole theory" of liquid predicts the follow ing relationship between viscosity and pressure. $\mu_p = \mu_{po} e^{b(p-po)}$ Where μ_p and μ_{po} are respectively, the viscosities at pressure p and atmospheric pressure po.

According to this theory,b is the pressure coefficient which is propotional to the hole volume V_H and inversely proportional to absolute temperature: $b = \frac{V_H}{RT}$ Pressure coefficients for polymer melts were measured by Semjonow and Westover, [3]who developed and build special capillary rehometer to measure this effect.

Chapter 3

HARDWARE DESIGN

Before designing the hardware interfacing , we need to know the form of the available data. It can be either in the analog form or in the digital form. Analog meters on the control penal gives the output in analog form. Since most computers are made up of digital circuitry it is required to convert the analog form of data in the digital. Following circuitry can help us designing hardware interface.

3.1 Logical Gates

Logical Gates are helpful to design logical circuits . Output of the gate depends upon the combination of the inputs. It can be either logic 'High' or logic 'Low'. There are different types of logic gates which can be used as required. The output of gates according to different input combinations are mentioned in the table, which is also called truth table of the gate.

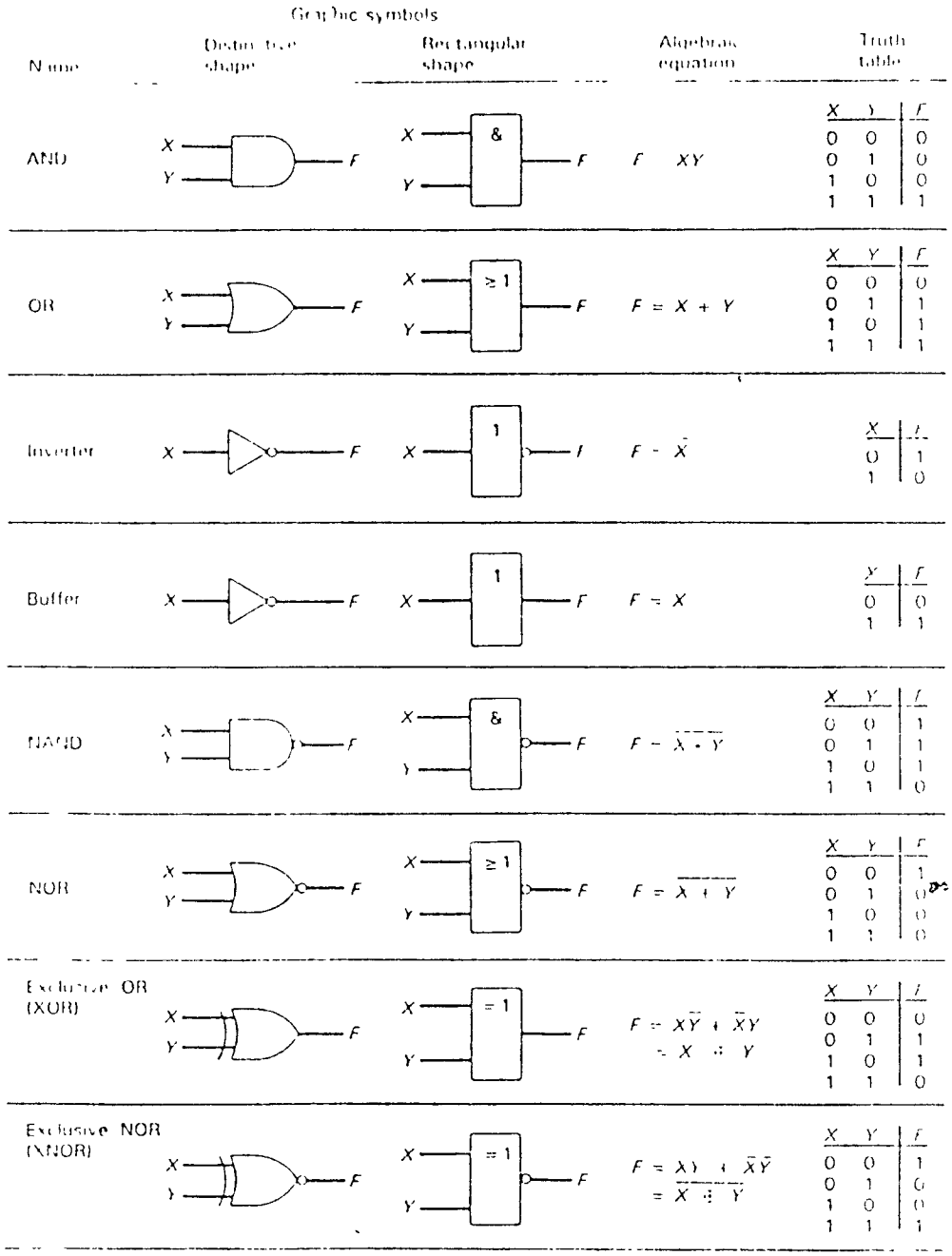


Figure 3.1: Digital Logic Gates

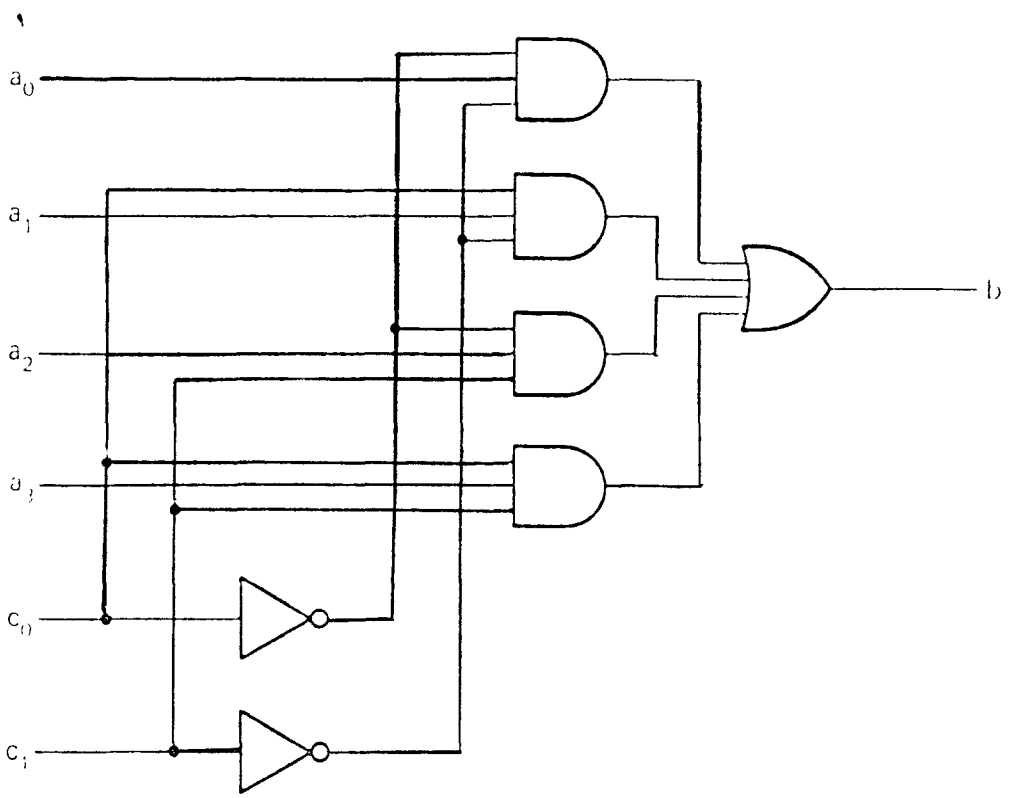


Figure 3.2: 4 to 1 line Multiplexer

3.2 Line Multiplexer

Line multiplexer is used when we have data coming up from number of lines and going to same or different destinations. Line multiplexer allows us to select one line at a time and sends the data to particular destination. Hence if we have different data (e.g. temp. of zone 1, zone 2 , ..etc) line multiplexer allows cpu to select one line at a time.

3.3 Analog to Digital Converter

As mentioned earlier it is important for us to convert analog form of data to digital in order to make them compatible with digital equipments. Circuit mentioned here uses compares the input voltage level with the reference voltage and converts them in either binary zero or one. Which is in turn either logic high or low. The graphical representation of voltage levels at the end of each clock pulse explains the conversion. The method mentioned here is of the successive approximation method, in which the digital output will be approximated by comparator circuitry with the help of the reference voltage.

3.4 Phase Locked Loop

Function of this circuitry is to give constant output frequency . It contains a filter , frequency divider and a voltage controlled oscillator. (VCO) Output of VCO depends upon input voltage. Whenever input voltage of VCO changes , the output frequency will be changed. The output of the VCO is fed beck to the frequency divider which changes the input to the comparator circuit and the input to the

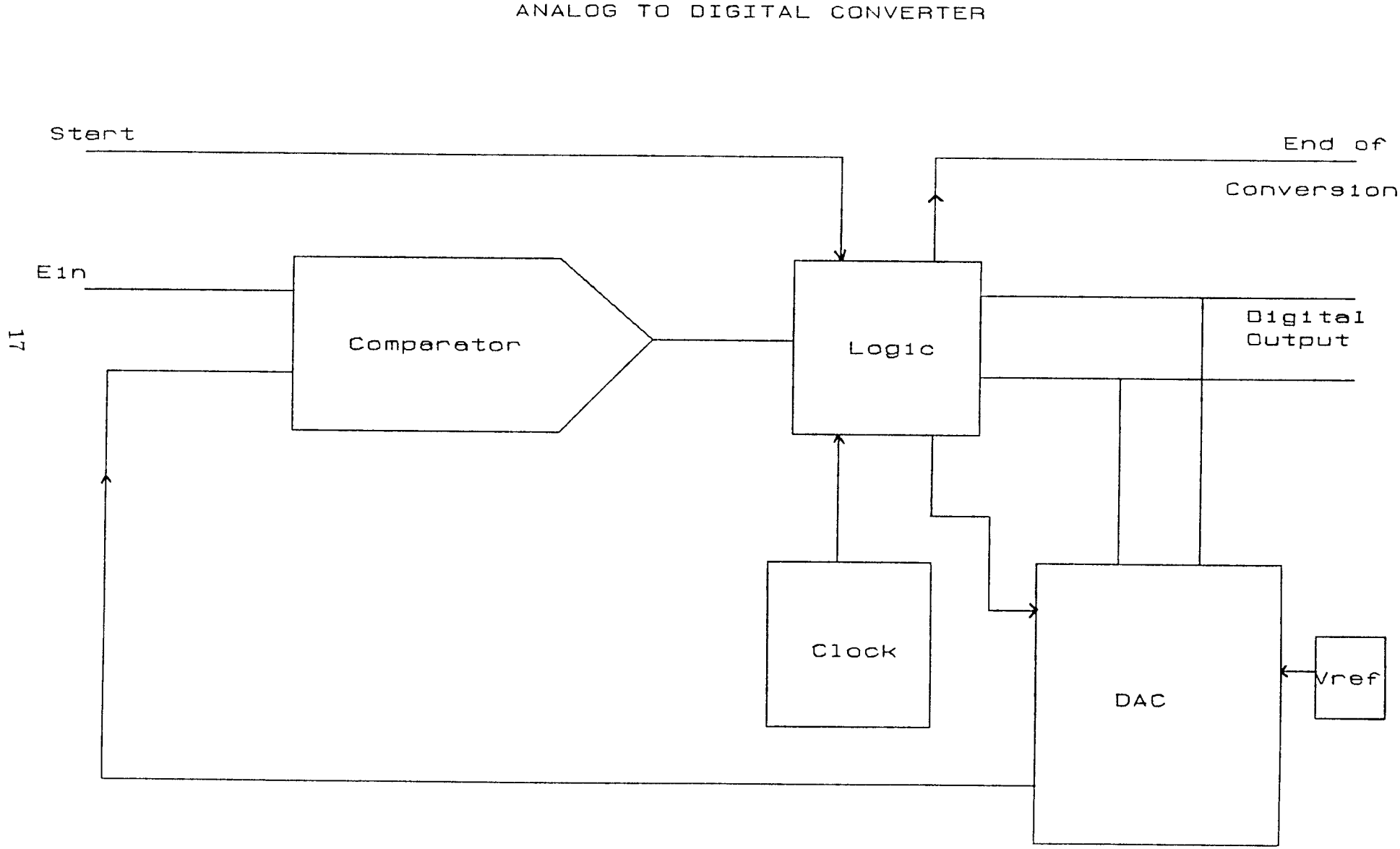


Figure 3.3 Analog to Digital Converter

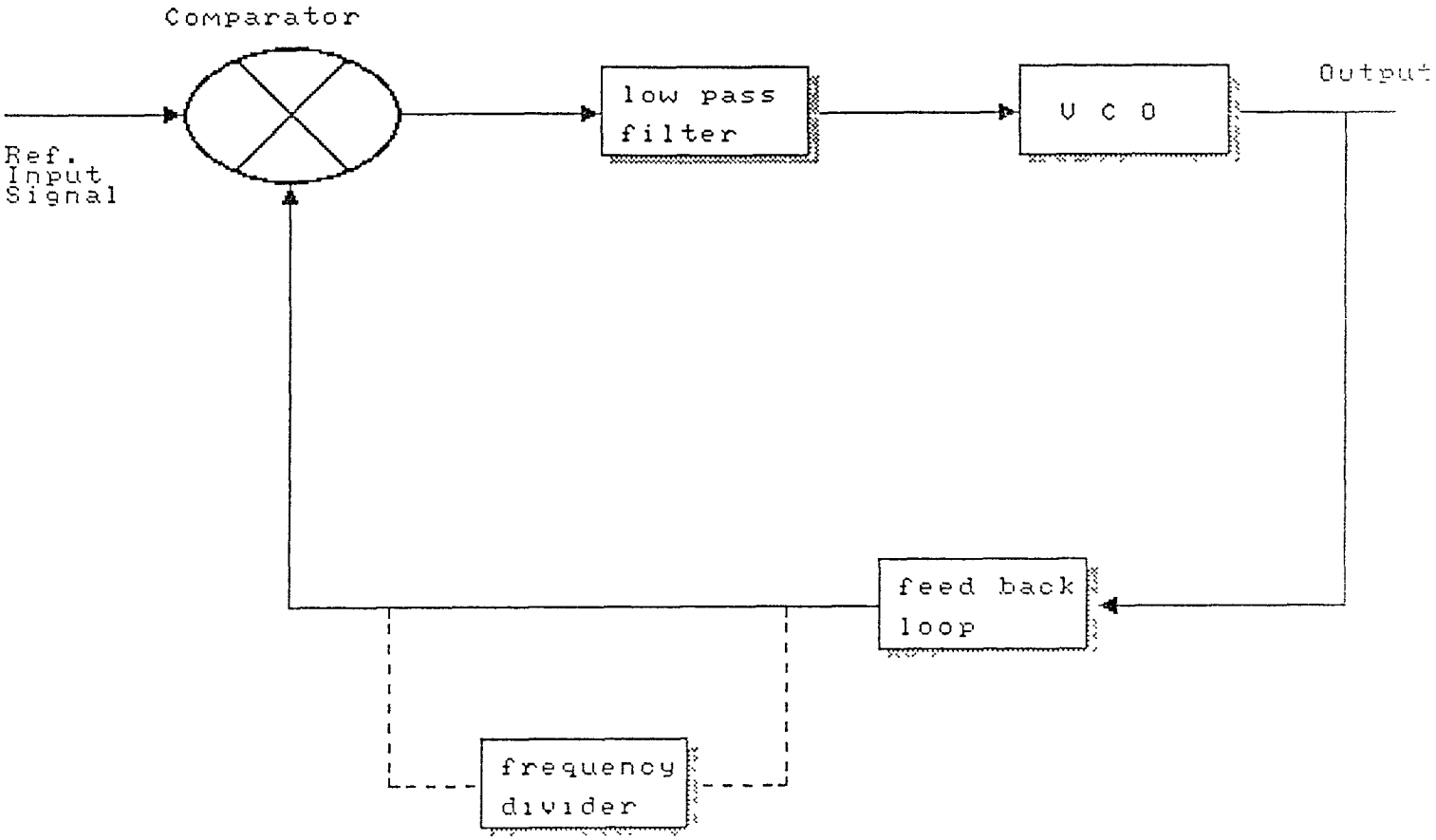


Figure 3.4: Phase Locked Loop

VCO remains unchanged. Phase locked loop circuit is designed to give the constant output frequency with the given reference input voltage. Hence the total analog to digital conversion will take place as follows.

1) Analog data will be filtered using lowpass or highpass filter (depending upon the input frequency) and will be fed to the multiplexer.

2) Output of multiplexer will be given to an analog to digital converter .

3) Analog to digital converter will be triggered at the fixed interval (whose frequency is determined by PLL) and converted digital data will be fed to the system bus.

3.5 Universal Synchronous Asynchronous Receiver Transmitter

USART is an acronym for Universal Synchronous/Asynchronous Receiver/ Transmitter. Early personal computer systems used software method for serial communications. The trouble is that it really ties up cpu and prevents the use of higher speed transmissions. USART was developed to simultaneously transmit and receive serial data , performing the appropriate parallel/serial conversion and inserting and checking the extra bits used to keep the serial data synchronized. The 8251A (USART) is used as a peripheral device and is programmed by CPU to operate using virtually any serial data transmission technique presently in use[5]. The USART accepts data characters from CPU in parallel format and then converts them in to a continuous serial datastream for transmission. Simultaneously, it can receive a serial data stream and convert them in to parallel data characters for the CPU. The USART will signal the CPU whenever it has received a character for the CPU or whenever it can accept the new character for the transmission. The CPU can read the complete status of USART at any time. This includes data transmission errors

and control signals such as SYNDET, TxEMPTY.

3.5.1 Features and Enhancements

The 8251A (USART) was considered for the design purpose because of following features. 8251A is a standered design of industry standered USART, the intel 8251. The 8251A operates with extended range of the microprocessors, and also maintains compatibilty with 8251. The 8251A incorporates all the key features of the 8251 and has the following additional features and enhancements:

- 8251A has double buffered data paths with separate I/O registers for control, status, Data In, Data out which considerably simplifies control programing and minimizes CPU overhead.
- In asynchronous operations, receiver detects and handles 'break' automatically,reliving the CPU of this task.
- A refined Rx initialization prevents the receiver from starting when in 'break' state,preventing unwanted intrrupts from a disconnected USART.
- At the conclusion of a transmission, TxD line will always return to the marking state unless SBRK is programmed.
- Tx Enable logic enhancement prevents a Tx Disable command from halting transmission until all data previously written has been transmitted. The logic also prevents the transmitter from turning off in the middle of a word.
- When External Sync Detect is programmed, Internal Sync Detect is disabled,and External Sync Detect status is provided via a flip–flop which clears itself upon a status read.

- As long as 8251A is not selected, the RD and the WR donot affect the internal operation of the device.

- The 8251A status can be read at any time but the status update will be inhibited during status read.

3.5.2 8251A Basic Functional Description

Like other I/O devices in a microcomputer system, 8251A's functional configuration is programmed by the system software for maximum flexibiliy.

In a communication environment an interface device must convert parallel format system data in to serial format for transmission and convert in coming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear 'transparent' to the CPU, a simple input or output byte oriented system data.

- Data Bus Buffer This 3-state,bidirectional, 8-bit buffer is used to interface the 8251A to system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the CPU. Control words, Command words and the status information are also transffered through Data Bus Buffer. The Command status and data in and data out are separate 8 bit registers to provide double buffering. This functional block accepts input from the system control bus and generates control signals for overall device operation. It contains Control word register and Command word register that store various control formats for the device functional definition.

- Read/Write Control Logic This block contains Pins which informs 8251A whether CPU is writing/reading data or Status information. Reset will put

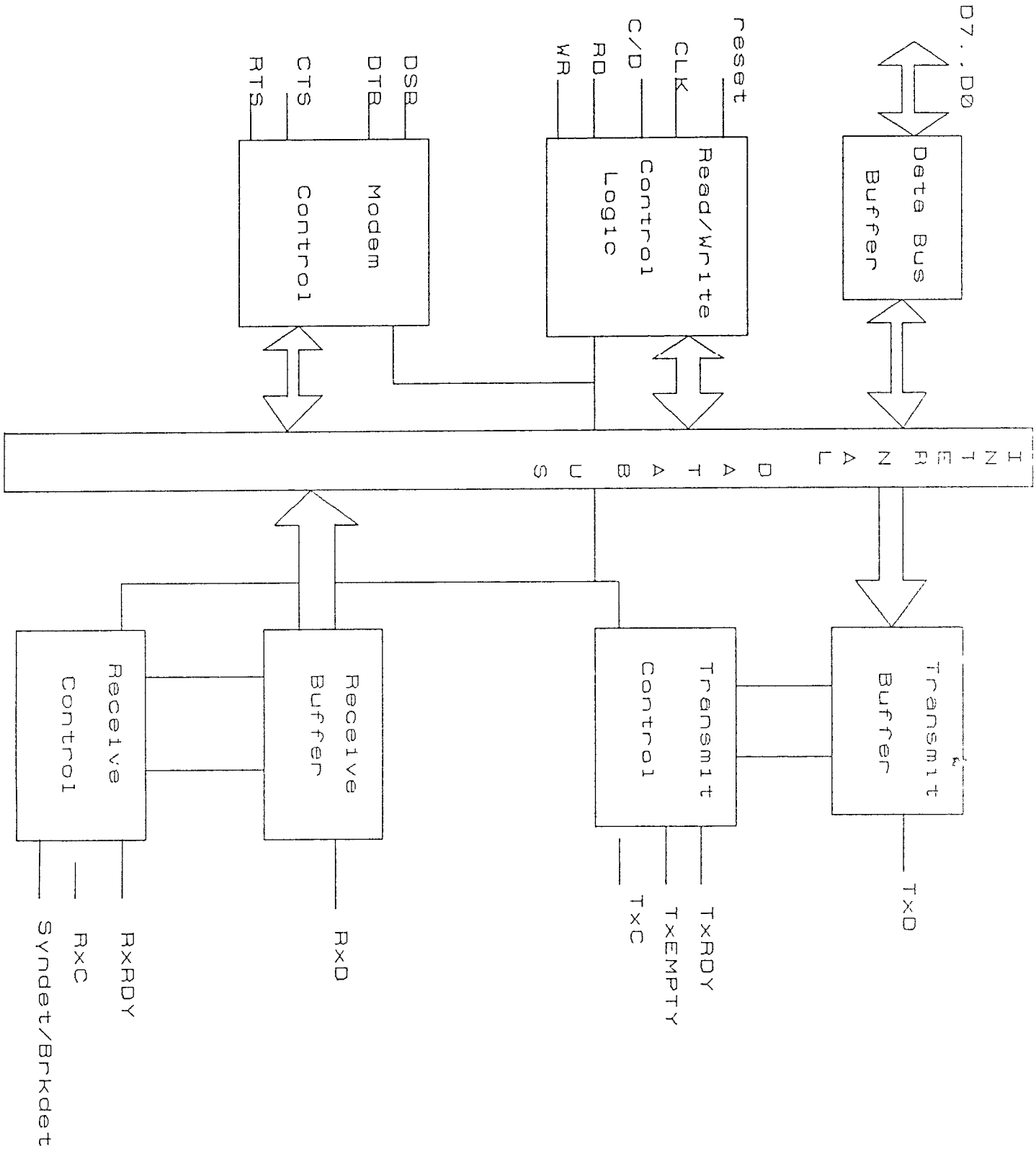


Figure 3.5: Functional diagram of 8251A

8251A in "idle" mode until next control word is written by CPU. Chip Select $\overline{CS}$ will select 8251A. No reading or writing will occur unless the chip is being selected.

- Modem Control: The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any Modem. The Modem control functions are general purpose in nature and can be used for functions other than Modem control.

- Transmitter Buffer: The Transmitter Buffer accepts parallel data from the Data Bus Buffer converts it to a serial stream, inserts appropriate characters and bits and outputs a composite serial stream of data.

- Transmitter Control: The transmitter control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

- Receiver Buffer: The Receiver accepts the serial data, converts this serial input to parallel format, checks the bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU.

- Receiver Control: This functional block manages all receiver related activities, issues and accepts signals to accomplish this function.

3.6 RS-232

The most important interfacing device is RS 232 which is a 25 pin serial port package. This port is used to transfer the data from device to the computer and from computer to the device. RS- 232 is a standard set by Electronic Industries Association (EIA) and works on negative logic levels. It gives larger voltage swings and

more noise-immune than conventional TTL. In terms of data communication, the device which behaves like a terminal (Extruder in this case) is called Data Terminal Equipment (DTE) and the device which behaves like a computer (computer in our case) is called Data Communication Equipment. (DCE)
The characteristics of RS-232 are as below:

- Maximum cable length 50 feet.
- Maximum data rate 20 kbps
- Maximum output current - 500ma to +500 ma.
- Maximum output voltage -15 or - 5 volts to +15 or +5 volts.
- Direct output resistance 300 ohms
- Form of operation is single handed

It has 25 signal lines, but most of the computers supports a set of seven. They are

- T * DATA: A serial data output from terminal (DTE) to modem (DEC). The line / signal through which the terminal sends data to modem.

- R * DATA: Control output from terminal DTE to DCE. The line/signal through which terminal requests permission to transmit the data to the DCE.

- RTS: Assertion of RTS instructs the DCE to receive data from DTE and to signify that is ready to receive by asserting CTS.

- CTS:Control input from terminal to DCE acknowledges the acceptance of terminal request to send data.

- DSR : Control input to the terminal DTE from DCE. The line through which DEC indicates whether data may be sent. DSR is not used in making decision.

- DTR: Control output from terminal to modem (DCE). The line through which terminal DTE indicates whether its on-line , inservice or active status.

- DCD: Control input from terminal to modem (DTE to DCE) . The line/signal through which modem (DCE) indicates that the communication channel to which DCE interface the other nonterminal side of DCE is an acceptable active state. This signal has meaning only in a communication context. DCD is off when no signal is being received or received signal is unsuitable for demodulation.

The general rule may be established as follow. When cross connecting between the devices make sure that output signal of one goes to the input of the another and viceversa.

pin no.	Common	RS 232	Description
1		AA	protective ground
2	Txd	BA	Transmit Data
3	Rxd	BB	Receive Data
4	Rts	CA	Request to send
5	Cts	CB	Clear to send
6	Dsr	CC	Data set ready
7	Gnd	AB	Signal ground
8	Cd	CF	Received line Signal Detector
9		-	Reserved for data testing
10		-	Reserved for data testing
11			Un assigned
12		SCF	Secondary received line signal detector
13		SCB	Secondary clear to send
14		SBA	Secondary Transmitted data
15		DB	Transmission signal element timing
16		SBB	Secondary Received Data
17		DD	Receiver Signal element timing
18			Unassigned
19		SCA	Secondary Request to send
20	Dtr	CD	Data terminal ready
21		CG	Signal Quality detector
22		CE	Ring Detector
23		CH/CI	Data signal rate selector
24		DA	Transmit Signal element timing
25			un assigned

Another important chips we need to know about are MC 1488 and MC 1489. MC 1488 converts TTL signals to RS 232 signals and MC 1489 converts RS232 signals back to TTL signals.

3.7 Physical Interface

The block diagram of the hardware interfacing of the instrument is as shown below. The functional operation of the device centers on the single chip instrument controller. e.g. of this could be given as INTEL 8748 microprocessor which has 2 k

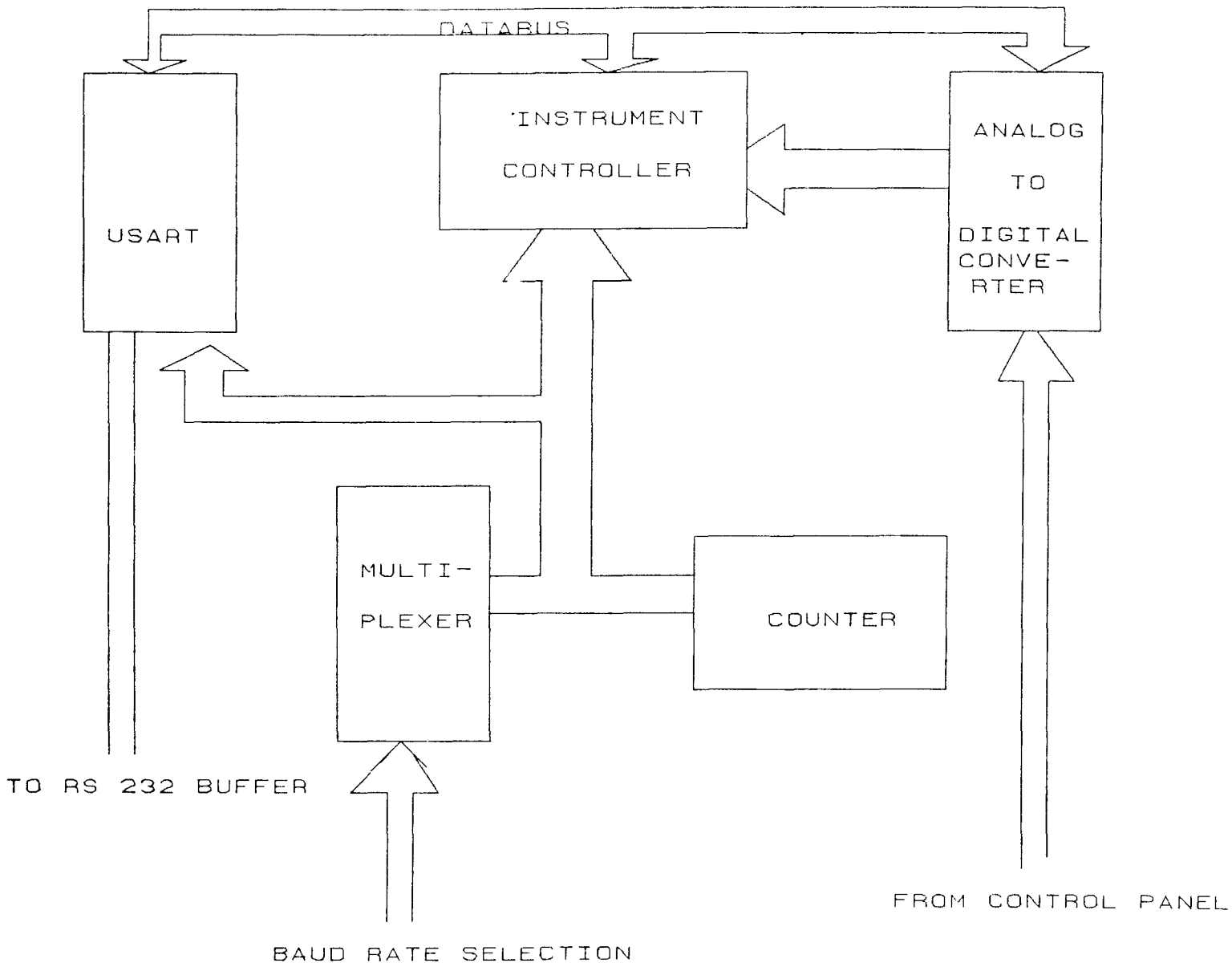


Figure 3.6: Block Diagram

Byte of EPROM (Electrically Programmable Read Only Memory) and 64 K byte of RAM on board and 8 bit timer installed. Personal Computer is connected to the device through RS232 and which is connected to the USART. The data conversion begins when the USART receives an ASCII character to initiate the data conversion. A/D converter will start receiving data from the control panel and will transfer the converted data to the data bus. Instrument controller will be programmed to stop the data acquisition after receiving a set of data. i.e. It will put the A/D converter to hold mode and transfer the data to USART. Program will append linefeed characters and carriage return characters along with ASCII characters. High level program demonstrated here will convert the ASCII character to its decimal mode and will store the data to the data base. The counter is used to synchronize all the sequence of operations and multiplexer could be used to select the different baud rates.

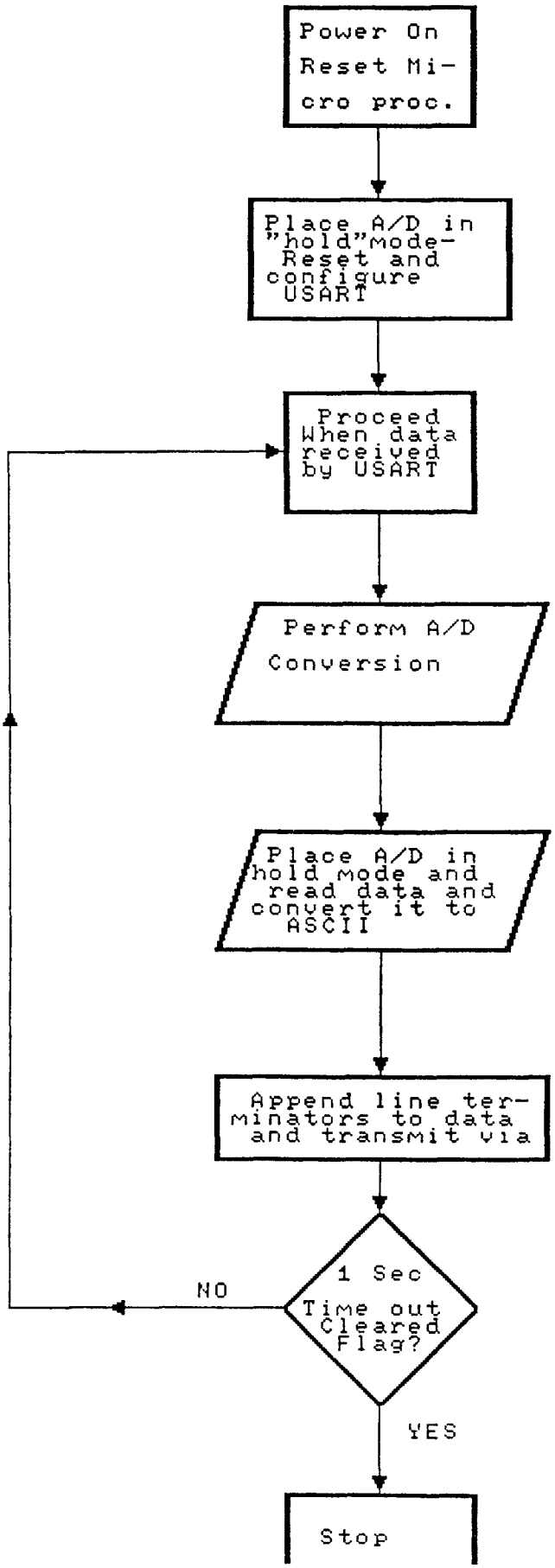
The assembly language program could be written in the particular processor's assembly language. The sequence of operations should be as following.

Immediately after power up the microprocessor configures the I/O ports for input or output,enables the the clock and resets the USART. The A/D will be in the "hold" mode at this stage. Microprocessor will determine the status of 8748's pins ,after determining the status, it will generate two control words, which are written to USART via databus. These control words will configure the USART for asynchronous communications with appropriate combinations of data,stop and parity bits.

An A/D conversion will begin when USART receives an ASCII character [1] , the arrival of which is detected by polling the RXRDY flag in USART with status register. The A/D converter is then released from 'hold' mode and its status output

[1]This character will be generated by user when user selects the option to start data acquisition

Figure 3.7: Assembly Language Flow Diagram



is polled until a high to low transmission is detected, indicating that the valid data from most recent conversions are available. The data are read by microprocessor and a binary to integer conversion is performed, with the resultant numbers than being converted to ASCII format. Following this leading zeros should be suppressed and should be loaded in to USART for transmission. The ASCII charecters of "carriage - return " and "linefeed" are appended to the data to complete transmission sequence. During the data transmission, the TXRDY flag of the USART status register is polled to provide "ready for next byte" signal. Following the transmission of data a 1 second time out period is generated after which RXRDY flag of the USART status register is reset.

Chapter 4

SOFTWARE DESIGN

4.1 Software Requirement Characteristics

These requirements are the basis for all software development activities. Before the plans for the projects are solidified, the technical interface and performance requirements for the software must be allocated by the program systems engineering organization. These requirements should define clearly, in a non ambiguous, traceable and a testable fashion, what the software must do to satisfy system requirements. Specification of the software requirement is the first technical task of the software development activity. The requirements are the basic specification of what the integrated software must do to satisfy the allocated requirements of the system.

The methodologies used to define and document the software requirements must interface with those used to specify the system level requirements. These methodologies results in an allocated set of system requirements upon which the functional design may be initiated.

The linking of these requirements to lower and higher level requirements is

an ongoing task which must be an integral parts of the methodologies selected and used by the projects.

4.1.1 Functional Design

Functional design is an orderly process which decomposes the functional definition of the system in to lower and lower level until the final level is reached and the software system design can proceed. There are several key elements which must be considered in definition and allocation of the functional requirements to system design components. These are,

- **Requirements Traceability** By decomposing software functional requirements from the system requirements and interface definition complete and clear Traceability is maintained back to the specified needs of the user.

- **Functional allocation** By Defining the top level flow of system support, and vigorously decomposing each of the elements which comprise the support to lower function levels, a system designer assures that what specifies as function system requirements accurately reflects the top level design of the system as well as the needs and requirements of the user.

- **Data Collection and Control** By vigorously maintaining a dictionary of system data elements, keyed to the functional allocation and decomposition process, a complete definition of internal system data requirements is maintained as design processes.

The purpose of the functional decomposition process is to successively break the individual software subsystems in to successively lower levels of details, adding essential derived and functional requirements as the decomposition proceeds. The lowest level of abstraction represents the basic division of subsystem in to separate loadable components which may then be described in program design language or some other form of representation. This decomposition process ensures that a system perspective is retained throughout the functional definition of the software and that requirement traceability is maintained throughout all levels of functional design.

4.2 Program Design

The function of the design technique is the following three step procedure.

- Consider problem environment and record our understanding of it by defining structures for the data to be processed.
- Form a program structure based on the data structure.
- Define the task to be performed in terms of the elementary operations available, and allocate each of those operations to suitable components of the program structure.

When the program structure is formed from the data structure, it is only skeleton. We will usually need to add some components when we come to allocate executable operations. If the task is entirely trivial, we may be able to express it directly in primitive operations. But a realistic program may contain hundreds or thousands

of instructions and cannot be written down directly, the task is too large and complex to be held, all at one time, in a human mind. The way of tackling this task is hierarchical structuring a program. Suppose if we wish to create a program P which, when executed performs the desired task T, we decompose T in to a number of sub tasks say T1, T2, T3 and we make corresponding decomposition in program P say P1, P2, P3. When P1 is executed it performs T1 and so on. So we now have to create four components P1, P2, P3 and P. P is needed to bring P1, P2 and P3 in to the correct relationship with each other. But P1, P2, P3 is probably too complex to be written down directly in the programming language and, we apply the same technique of decomposition to them in turn. We decompose T1 in to sub task say, T11, T12, T13 and we decompose P1 correspondingly. The process continues level by level until no component remains which is too complex to be coded directly in the programming language as a single executable instruction.

4.3 Design of an Efficient Software

Following are the tools to design an efficient software. It should have the following properties.

1) Be of low complexity.

2) Be easily understood.

3) Be easily documented.

4) Have a high probability of being correct.

5) Be easily modified and maintained.

To develop above mentioned properties, the methodology should include following

1) Top down Design

2) Modularity

3) Design Language

4) Documentation

4.3.1 Top Down Design

The design should be partitioned in to different functional levels. At the top levels, which is completed first, the design should be most general and at the bottom level it should be most detailed. This approach allows design to proceed from the abstract at the top to the specific at the bottom in a natural systematic way.

4.3.2 Modularity

All the system functions which belong together or which manipulate a common data structure should be placed in to a single module. This provides a way of systematically design a system. It also provides a control of access to each data structure since only the procedures which are in a particular module have access to the data structure within that module. Each module should be broken down in to procedures so that each procedures implements only a single function, if possible.

4.3.3 Design Language

The design language should consist of natural language which includes a small number of 'programming construct' such as DO, DO WHILE, FOR i = 1 to N Do ....etc. The design language also have provision for specifying inter procedure parameters clearly and un ambiguously.

4.3.4 Documentation

The design language version of the system design provides a high degree of visibility in to the system. When combine with the module documentation and additional information,the design language should provide a base for the next level of documentation.

There are two types of software design necessary to complete an interfacing requirements

4.4 Assembly language program

Which enables the communication of data between DCE and DTE and stores it in the memory of the computer. It can be written in the assembly language of the processor. The assembly language program used for plasticating extruder was developed in assembly language of Intel 8086 processor, and is discussed in detail in this section. Appendix A gives the whole listing of the program. It is however important to mention here that the module developed here is only the part of the complete data acquisition requirements, and could be used for loading and executing a file. This could be used when a user wants to communicate with the processor. The execution procedure is as follows. The program begins by printing a message on the screen "program starting". The next section of the code (starting at ASSUME DS:PREFIX) does the setup for loading a file from the disk. What we need to find is the paragraph number at the end of the program. A paragraph is 16 bytes. Hence the paragraph number is the memory address divided by 16, in other words, the segment address. To get this, the end address of the current program is obtained by loading the offset address of eop+10h. This is divided by

16 and added to the program segment address to give the paragraph number of the end of the program. The paragraph numberof the program prefix segment is then subtracted from the program and end paragraph number to get the total program length in paragraphs in BX register. Making sure that ES points to the starting segment of the program (the program prefix), DOS function 4A is called to free all memory above the program. This is necessary because the DOS internally marks all the memory as being allocated to a program when it is loaded in to memory.

Next four quantities are initialized in a parameter block (PBLOCK) that's been setup in the data segment: the segment address of the environment string, a DWORD pointer (offset and segment) to a command line and DWORD pointers to the two file control blocks (FCB'S) in the original program prefix (ENVSEG). Since the default value of the environment are used here, the environment address is just picked up from the original program prefix(ENVSEG). Similarly, the program uses the original command line and two default FCB setups in the original program prefix so, pointers to them are stored in PBLOCK. Since the offsets of these three quantities in the program prefix are known in advance, they are fixed in the definition of PBLOCK, and only the segment address of the original program need to be filled in. Now ES:BX is pointed to the beginning of the parameter block, and DS:DX is pointed to the name of the file to be loaded and executed. Unless the file name is preceded by the drive specification, the computer looks for the file on default drive only. In the example file name is given as "asc-file", we can change this to whatever file we want to load. The file name is given by an "ASCIIZ" string, which is just an ASCII string followed by a byte of zero. Finally, the current value of SS:SP is saved in the data segment. Everything is now set up to load and execute the new program using DOS function call 4BH (set AH=4BH and AL= 0). When this function call is executed, "asc-file" is loaded in the memory and executed.

When "asc-file" is executed, control is automatically passed to the next instruction in the program following the INT 21H (that is, to MOV AX,DSEG). Since all registers are destroyed by function call 4BH, the first action must be to restore DS so it points to the data area again, and then to reload the stack pointer. The program then finishes by printing "program completed" message.

One limitation of this program is that you cannot use it load a program and modify it before execution.

In order to write to the file we need to develop an assembly language module which can create a file containing ASCII characters, so that a program demonstrated here can read the file and perform required operations. The explanation of the assembly language program module 2 is given below. The program begins by creating file whose name is MYFILE.TXT. This is done by pointing DS:DX at the file name, putting the desired file attribute in CX, and calling DOS function 3CH. The file name must be an ASCIIZ string - that is, a string of ASCII characters ending with a byte of 00. If you wish, you can optionally precede the file name with a drive specification such as "A:". If no drive is specified, as is done in this example program, the current default drive is assumed. Also, if you're using a hard disk system with subdirectories, You must specify a path name in front of the file name just as you do to specify a file when you're in the operating system.

The file attribute byte is 00 for normal files, so it is set to 0 in the program. If you want to create a special type of file such as a read-only file, a system file, or a hidden file, you must change this value accordingly. DOS function call 43H can also be used to change the file attribute byte.

When the file is created by function call 3CH, It's automatically opened fo for either reading or writing, and the file handle is returned in AX. If no other disk files are open, you normally get a file handle of 5.

Immediately after the function call that creates the file, the program checks the carry flag and jumps to an error routine if it is set. You should always make this check immediately after the Dos function call that creates or opens a file. When an error occurs during a DOS file handling call, it returns with carry flag set and an error code in AX If you simply ignore the error and continue, you'll be using the error code as the file handle. This is likely to produce behaviour that is at best unpredictable, and worst disastrous.

The error routine used in the example is very minimal. It simply announces that an error has occured and quits. A more sophisticated routine would examine AX to find out what kind of error it was and then send the appropriate message and/or take other appropriate recovery actions.

The next section of code writes the contents of BUFR into the newly created file. All that's required to do this is to point DS:DX at the beginning address of BUFR, put the number of cbytes to write in CX, and have the file handle in BX. DOS function call 40H then takes care of everything else. Although it has not been done in the skeletal routine here, you should also follow all disk write function calls with a check for errors and an appropriate error handling routine. This is especially important if the data being written to disk is valuable or not easily replaceable. In a good file-handling assembly program, at least half the code should probably be devoted to error handling.

The program ends by closing the file with a call to DOS function 3EH. All that needs to be done here is put the file handle in BX and make the call. It's vital that the file be explicitly closed with the call before the program ends; otherwise you'll lose some or all of the data. The reason is that when you write to a disk file with DOS function 40H, the data is first written into a 512-byte buffer (the size of one disk sector) in memory. Only when a full 512 bytes have been written into the

buffers: is the data actually written onto the disk. This speeds up disk operations tremendously, but unless you've written an exact multiple of 512 bytes, part of the disk file remains in memory until a close file command is executed.

4.5 High level Program

This reads stored data and converts it in the form of an information. It requires two settings.

1) Setup: Which is nothing but selection of menu and display of the required information.

2) Acquire: This is very important part of the design . This acquires data from the process at intermediate levels and stores it and retain the address of the stored data. Also it performs the required transformation convert the data in to information. When ever the process ends it closes all the files and concludes the run of the software.

The source code of the program is mentioned in the Appendix. The explanation of the program is given in detail in this section.

The program starts with the menu asking user to start data acquisition or to quit. Upon selection of 1 or 2 it goes to particular option. If option 1 has been selected than, it will display another menu and ask for the choice. If user wants to see the format of ASCII character,which (output from assembly language program) the program will accept, it will show the file 'Test.data', in which data for demonstration purpose has been stored. Choice no. 2 is to convert the ASCII characters stored in to 'Test.data' and will write to a file 'Out.Data'. The file Out.data is in its append mode. Hence each time the program runs, the output will be appended and not be overwritten.

The procedure "re-ad()", and "pr-int()" are used to put the converted output

in formatted form. The "re-ad()" will read the data form the out.data, and will allocate the memory according to the specified structure and store the data in a link list. "Pr-int()" will read the list and put the formatted output in the file 'out1.data.' The limitation here is that, the input data has to be in the specified format. If the user has a specific file that is to be converted in to ASCII, the program gives the option to enter the file name. It reads the file using file pointers and than stores the converted output to the file named "asc-flie".

Chapter 5

RESULTS

The total data acquisition process is explained as follows. The output of the control panel data will be transmitted in terms of electrical signals to the instrument. USART on the instrument will receive the data and transmit it to the personal computer via RS 232 protocols. Functional details are given in Chapter 3 (section 3.5, 3.6).

An assembly language program will control whole sequence of operations. It will convert received electrical signals to ASCII format and will append line feed characters to it. Flow diagram of the program is mentioned in fig 3.7. Two modules of assembly language program for loading and executing the file and to write the data in the disk are developed using 8086 processors assembly language and are listed in the appendix. Explanation of the modules are given in section 4.4. The output of the assembly language will be in ASCII string format, which is nothing but the data received from the control panel of the extruder machine. One example of output of assembly language program is mentioned below. The computer program was developed to acquire data. Input to the program was given in the ASCII format

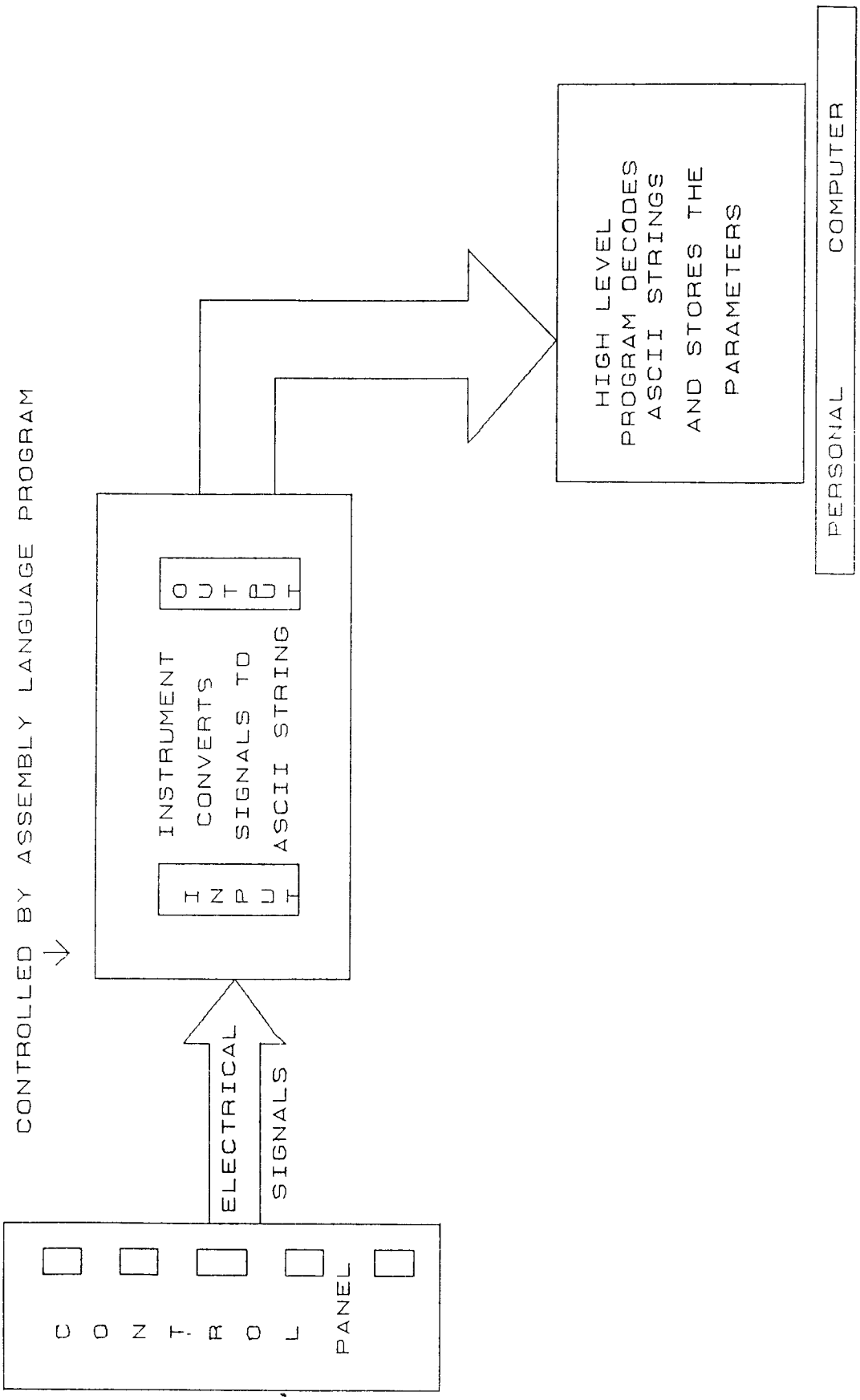


Figure 5.1: Block Diagram of Data Acquisition Process.

```
/*************************************************************/
/              Example Of ASCII string                      /
/              Input to the Program                         /
/              Output of the assembly language program      /
/*************************************************************/



505332324846483232565646533232484648323248464832325054 48
324846483248464832535148324846483248464832535746483248 46
483248464832525746483248464810534832484648324953573248 46
483248464832534848324846483248464832555453324846483248 46
483249485546533248464832484648325657465332484648105553 32
484648324957563248464832484648325454483248464832484648 32
565648324846483248464832495250324846483248464832495048 46
483248464810494848324846483250504832484648324846483256 49
483248464832484648325750483248464832484648324955523248 46
483248464832495252464832484648105053324846555232574946 48
324951464832514652543249575632495746533250465557325253 53
324950465332483248464832484648324846555332525646483250 49
465310534832494650573249525732495146563256464853325157 48
325048464832534655533254534832495246533232483248324832 49
465249325656464832504946531055533249465451324956523249 53
464832495246573253515332505046553257464853325551483249 56
464832324832324832484648324946575632494951465332504946 53
104948483249465756325048573249534653325052465032545053 32
505246483249504653325756483249574650324832483248324832 50 46
535332495155325049465310505332554648533255504832495246 48
324949465232534932495746523249494655324950533249514648 32
554651543251544648325049465310544649533250574653325049 46
531053483249504656324948553249544648325052465632494856 32
504846483250504648324956553249534649324952465432545346 53
325050464832494946521053514653325049465310555332515646 48
325749464832495646503252524648325555325049465432525046 55
324950493249564648325155464932535146483250514648325057 46
503210525246533250504653104948483253504650324948513250 48
465332535746483257533250504648325355465432495148324957 46
533253504648325551465232505246483252484653325354464810 50
514652103210-1
```

and in a file called TEST.DATA. The format was specified according to the output of the assembly language, and hardware design. Output is obtained in an output file. The name of the output file is specified as OUT.DATA and tabulated output is kept in a file called OUT2.DATA. The program was tested with two sets of data. During the first run of data , it creates two output files,out.data and out2.data. During the second run it appends both files, and does not overwrite them. In case of missing data it gives an error message. Input data is in the form of ASCII string with line feed character appended to it, and the output of the program is mentioned here.

Where R.P.M.(revolutions per miniute) is the number of screw rotations per minute.

Rate1a – is the flow rate of the polymers in zone 1 A.

Pres1a – is the pressure indicated on the control panel for Zone 1 A.

Temp1a – is the temprature of the mold in zone 1 A.

Similarly , Rate1b is the flow rate of the polymers in zone 1 B. and so on.

Readings are taken at the fixed interval of screw rotations. Functional details of the program are given in the section 4.5.

OUTPUT OF THE PROGRAM

DATA SET # 1
(OUT2.DATA)

R.P.M	25.00	50.00	75.00	100.00
Rate1a	0.00	0.00	0.00	0.00
Prs1a	88.50	159.00	198.00	220.00
Temp1a	0.00	0.00	0.00	0.00
Rate1b	0.00	0.00	0.00	0.00
Prs1b	260.00	500.00	660.00	810.00
Temp1b	0.00	0.00	0.00	0.00
Rate2	0.00	0.00	0.00	0.00
Prs2	530.00	765.00	880.00	920.00
Temp2	0.00	0.00	0.00	0.00
Rate3a	0.00	0.00	0.00	0.00
Prs3a	59.00	107.50	142.00	174.00
Temp3a	0.00	0.00	0.00	0.00
Rate3b	0.00	0.00	0.00	0.00
Prs3b	49.00	89.50	120.00	144.00
Temp3b	0.00	0.00	0.00	0.00

R.P.M	25.00	50.00	75.00	100.00
Rate1a	0.74	1.29	1.63	1.98
Prs1a	91.00	149.00	184.00	209.00
Temp1a	13.00	13.80	15.00	15.50
Rate1b	3.46	8.05	14.90	24.20
Prs1b	198.00	390.00	535.00	625.00
Temp1b	19.50	20.00	22.70	24.00
Rate2	2.79	5.75	9.05	12.50
Prs2	455.00	650.00	730.00	980.00
Temp2	12.50	14.50	18.00	19.20
Rate3a	0.00	0.00	0.00	0.00
Prs3a	0.00	0.00	0.00	0.00
Temp3a	0.00	0.00	0.00	0.00
Rate3b	0.75	1.41	1.98	2.55
Prs3b	48.00	88.00	113.50	137.00
Temp3b	21.50	21.50	21.50	21.50

R.P.M	25.00	50.00	75.00	100.00
Rate1a	7.05	12.80	38.00	52.20
Prs1a	720.00	107.00	91.00	103.00
Temp1a	14.00	16.00	18.20	20.50
Rate1b	11.40	24.80	44.00	59.00
Prs1b	51.00	108.00	77.00	95.00
Temp1b	19.40	20.00	21.60	22.00
Rate2	11.70	22.00	42.70	57.60
Prs2	125.00	187.00	121.00	130.00
Temp2	13.00	15.10	18.00	19.50
Rate3a	7.36	14.60	37.10	52.00
Prs3a	36.00	65.50	53.00	73.40
Temp3a	21.50	22.00	23.00	24.00
Rate3b	6.15	11.40	29.20	40.50
Prs3b	29.50	53.50	44.50	56.00
Temp3b	21.50	21.50	22.50	23.40

OUTPUT OF THE PROGRAM

DATA SET # 2
(OUT2.DATA)

R.P.M	25.00	50.00	75.00	100.00
Rate1a	0.00	0.00	0.00	0.00
Prs1a	88.50	159.00	198.00	220.00
Temp1a	0.00	0.00	0.00	0.00
Rate1b	0.00	0.00	0.00	0.00
Prs1b	260.00	500.00	660.00	810.00
Temp1b	0.00	0.00	0.00	0.00
Rate2	0.00	0.00	0.00	0.00
Prs2	530.00	765.00	880.00	920.00
Temp2	0.00	0.00	0.00	0.00
Rate3a	0.00	0.00	0.00	0.00
Prs3a	59.00	107.50	142.00	174.00
Temp3a	0.00	0.00	0.00	0.00
Rate3b	0.00	0.00	0.00	0.00
Prs3b	49.00	89.50	120.00	144.00
Temp3b	0.00	0.00	0.00	0.00

R.P.M	25.00	50.00	75.00	100.00
Rate1a	0.74	1.29	1.63	1.98
Prs1a	91.00	149.00	184.00	209.00
Temp1a	13.00	13.80	15.00	15.50
Rate1b	3.46	8.05	14.90	24.20
Prs1b	198.00	390.00	535.00	625.00
Temp1b	19.50	20.00	22.70	24.00
Rate2	2.79	5.75	9.05	12.50
Prs2	455.00	650.00	730.00	980.00
Temp2	12.50	14.50	18.00	19.20
Rate3a	0.00	0.00	0.00	0.00
Prs3a	0.00	0.00	0.00	0.00
Temp3a	0.00	0.00	0.00	0.00
Rate3b	0.75	1.41	1.98	2.55
Prs3b	48.00	88.00	113.50	137.00
Temp3b	21.50	21.50	21.50	21.50

R.P.M	25.00	50.00	75.00	100.00
Rate1a	7.05	12.80	38.00	52.20
Prs1a	720.00	107.00	91.00	103.00
Temp1a	14.00	16.00	18.20	20.50
Rate1b	11.40	24.80	44.00	59.00
Prs1b	51.00	108.00	77.00	95.00
Temp1b	19.40	20.00	21.60	22.00
Rate2	11.70	22.00	42.70	57.60
Prs2	125.00	187.00	121.00	130.00
Temp2	13.00	15.10	18.00	19.50
Rate3a	7.36	14.60	37.10	52.00
Prs3a	36.00	65.50	53.00	73.40
Temp3a	21.50	22.00	23.00	24.00
Rate3b	6.15	11.40	29.20	40.50
Prs3b	29.50	53.50	44.50	56.00
Temp3b	21.50	21.50	22.50	23.40

R.P.M	25.00	50.00	75.00	100.00
Rate1a	0.00	0.00	0.00	0.00
Prs1a	1380.00	1800.00	1970.00	2170.00
Temp1a	0.00	0.00	0.00	0.00
Rate1b	0.00	0.00	0.00	0.00
Prs1b	3000.00	3000.00	3000.00	3000.00
Temp1b	0.00	0.00	0.00	0.00
Rate2	0.00	0.00	0.00	0.00
Prs2	3000.00	3000.00	3000.00	3000.00
Temp2	0.00	0.00	0.00	0.00
Rate3a	0.00	0.00	0.00	0.00
Prs3a	3000.00	3000.00	3000.00	3000.00
Temp3a	0.00	0.00	0.00	0.00
Rate3b	0.00	0.00	0.00	0.00
Prs3b	2100.00	2750.00	2950.00	7300.00
Temp3b	0.00	0.00	0.00	0.00

R.P.M	50.00	75.00	100.00	50.00
Rate1a	16.10	26.40	36.70	35.00
Prs1a	1480.00	1610.00	1725.00	780.00
Temp1a	206.00	214.00	216.00	201.00
Rate1b	23.30	38.80	58.60	25.50
Prs1b	1620.00	1810.00	1975.00	675.00
Temp1b	205.00	208.00	216.00	200.00
Rate2	28.30	40.30	55.80	31.30
Prs2	1750.00	1850.00	2000.00	810.00
Temp2	205.00	210.00	215.00	204.00
Rate3a	27.60	40.70	57.00	38.60
Prs3a	1750.00	1900.00	2025.00	880.00
Temp3a	198.00	199.00	210.00	198.00
Rate3b	18.20	27.00	36.70	27.90
Prs3b	1520.00	1580.00	1690.00	750.00
Temp3b	204.00	210.00	215.00	194.00

R.P.M	75.00	100.00	50.00	75.00
Rate1a	51.00	69.30	42.00	61.10
Prs1a	910.00	1000.00	560.00	620.00
Temp1a	204.00	208.00	195.00	200.00
Rate1b	44.50	67.20	26.00	46.00
Prs1b	810.00	900.00	400.00	510.00
Temp1b	204.00	208.00	198.00	196.00
Rate2	46.00	64.00	32.80	51.50
Prs2	875.00	950.00	500.00	575.00
Temp2	204.00	206.00	200.00	204.00
Rate3a	56.00	78.80	42.00	62.80
Prs3a	975.00	1010.00	550.00	603.00
Temp3a	200.00	204.00	190.00	192.00
Rate3b	43.30	57.70	31.20	46.20
Prs3b	850.00	900.00	500.00	575.00
Temp3b	195.00	204.00	194.00	194.00

R.P.M	100.00	50.00	75.00	100.00
Rate1a	82.50	48.50	70.40	96.00
Prs1a	680.00	255.00	290.00	305.00
Temp1a	204.00	190.00	194.00	202.00
Rate1b	68.80	26.20	46.70	72.50
Prs1b	575.00	175.00	210.00	270.00
Temp1b	206.00	199.00	202.00	208.00
Rate2	70.00	36.30	52.30	73.00
Prs2	625.00	225.00	240.00	260.00
Temp2	204.00	196.00	200.00	206.00
Rate3a	85.00	44.00	66.00	87.40
Prs3a	680.00	270.00	330.00	350.00
Temp3a	194.00	190.00	196.00	202.00
Rate3b	63.70	36.90	53.40	70.70
Prs3b	625.00	240.00	255.00	280.00
Temp3b	198.00	190.00	194.00	198.00

R.P.M	25.00	50.00	75.00	100.00
Rate1a	28.00	57.50	86.30	112.00
Prs1a	0.00	0.00	0.00	0.00
Temp1a	186.00	185.00	186.00	190.00
Rate1b	12.30	28.30	47.00	75.00
Prs1b	0.00	0.00	0.00	0.00
Temp1b	192.00	195.00	198.00	200.00
Rate2	0.00	40.20	59.00	80.30
Prs2	0.00	0.00	0.00	0.00
Temp2	0.00	0.00	0.00	0.00
Rate3a	0.00	50.30	76.20	102.00
Prs3a	0.00	0.00	0.00	0.00
Temp3a	0.00	184.00	186.00	190.00
Rate3b	20.20	44.50	65.00	89.00
Prs3b	0.00	0.00	0.00	0.00
Temp3b	190.00	182.00	188.00	186.00

Chapter 6

DISCUSSION

6.1 Pressure and Temprature Effects

As mentioned in section 2.4 and 2.5, temperature and pressure have a significant effect on the final product in the extrusion process. The activation energy of a nonNewtonian fluid varies with temperature in addition to shear rate or shear stress. An increasing temperature causes a decrease in the activation energy and also tends to suppress nonNewtonian effects.

Also, according to Duvdevani and Klein[1] theory, 10,000 *psi* increase in the pressure will cause the viscosity to increase about 35%,which also can cause a significant effect on the output.Hence it very important to study the changes in the temperature and pressure while the process is running.

6.2 Use of program

The software program developed here will expect data from assembly language program in ASCII format. Since according to hardware design mentioned in chapter 3,

[1]References marked at the end of the text

the transmission of the data will be in the serial format , the output of the assembly language should append a linefeed character after each transmitted character. The sequence of operation will be as follows

When the user presses a key to start data acquisition, an ASCII character will be transmitted to the USART and the data conversion will start. Assembly language program will select proper channel and will start conversion. Details are mentioned in chapter 4. High level program demonstrated here will accept data characters in ASCII format and convert them in to readable decimal format. Parameters will be stored in a file called 'out.data'. Each time program runs , the file will be appended (example of which is given in the results) and the previous data will not be lost. If the user wants to communicate with the microprocessor, the program does have an option to convert a file to ASCII format, a small assembly language program routine mentioned in the appendix will transfer the file to the interfacing unit, thus allowing user to control operation through the computer. The program uses 'dynamic memory allocation' concept which is more advantegious compared to static memory allocation, as it does not have any space restriction and also there will not be any wastage of the space. Assembly language program mentioned here is written in the assembly language of Intel 8086 processor and can be used for file loading and execution.

Microprocessor based data acquisition not only saves manual key board entry, but also provides fast and permenant storage of information. It also provides an option to control the process from the key board.

Chapter 7

CONCLUSIONS

The following conclusions are made from thorough analysis of the material presented in the previous chapters.

(1) The microprocessor based software control system was developed for data acquisition. Suggested hardware design was developed using Intel 8086 microprocessor and software is developed in C. The module of assembly language program demonstrated here is written in the assembly language of 8086 processor. Data acquisition format was developed based on the data sheet given from 'Killion Extruders Inc.' Simulation of the system operation with instrument controller Intel 8748 and peripherals with associated software can be effectively used for software based data acquisition system as demonstrated for the plasticating extruders.

(2) The use of microprocessor based instrument allows the results of an A/D conversion to be transmitted as an ASCII number terminated by "carriage-return,line-feed". Data transmitted by the instrument are thus in the same format as those expected from a video terminal, allowing high-level language programming to be used for data reception.

(3) Also, problems which can arises when using unintelligent instruments, such as transmission of binary data which constitutes an ASCII control code – are avoided.
(4) The program mentioned in the appendix will receive an ascii string as an input, convert it in the form of information and will store the data in the files. Error message will be displayed where required.

Chapter 8

FUTURE WORK

Further enhancement of the thesis could be

(1) Interfacing the files to a database for permanent storage of the acquired data. The program listed in the appendix creates the file to store the data,but it does not store the data to a database.

(2) Two modules of assembly language program has been developed. It may be of further interest to develop other modules, to control the input output sequence and to control extrussion process through the keyboard.

(3) The block diagram designed here is for 8 channel data acquisition. It could be expanded to 16 channel data acquisition. For this, conversion sequence remains similar to one mentioned in ch.3. The major difference trigger character is assumed to be an ASCII no. between 0 and 15,and should be appropriately decoded on to the multiplexer channel,prior to releasing A/D converter from " hold" mode.

REFERENCES

[1] Michale W. Evans, " Productive software Test Management ," *A Wiley-Interscience Publications* NY, 1984

[2] C.M. Trotter, W.W. Carson, "Data Acquistion from a parallax bar via an RS232C interface," *Review of Scientific Instruments*, Nov 1985

[3] Zehav Tadmor, Imrich Klien," Engineering Principles of Plasticating Extrusion."*Robert E. Krieger Publishing Company* , 1978

[4] Archie J. Weith, Herbert Simonds, William Schack, "Extrusion of Plastics,Rubber and Metals," *Reinhold Publishing Corporation*, NY 1957

[5] N.J.Rao, " The Intel 8085/A Family Data Manual,"*Center for Electronics Design and Technology, IISC* Bangalore,India 1988

[6] Kim Jaeho, " Design of a Microcomputer controller for Mechanical Engineering applications," *Master's Thesis , NJIT* 1985.

[7] Murray Sargent III, Richard L. Shoemaker, "The IBM PC from the inside out," *Addision-Wesley Publishing Company* Aug. 1990.

[8] Len Egol, " PCs Simplify Data Collection And Management," *Chemical Engineering* pp 157-161 July 1990.

[9] W. Carlson, "Applications of data acquisition system," *Computers in Industry* Vol:13, Iss 1,pp 49-59, Sep 1989.

[**10**] M. Liu , " PC makes it easier to collect and analyse data," *Electronic Business* Vol:14, Iss 19, pp 115-116 , Oct. 1988.

[**11**] Milis A.J., " PC Data Acquisition," *Systems Inter national (UK)* Vol: 13 Iss 10, pp 35-36, oct. 1985.

[**12**] Rowan O'Riley,"PC not for just office anymore," *Production Engineering* Vol: 32, Iss 11, pp 60-69, Nov 1985.

[**13**] Keith R. Kreisher.,"The Screw-Design Debate: Flexibility or Job Dedication?,"*Modern Plastics* pp 60-61, April 1990.

[**14**] Jack K. Rogers.," Big parts blow molders grow more complex; more versatile," *Modern Plastics* pp 64-68,April 1990.

[**15**] Ayyagari A.J.," Simulation of a micro processor control led drill pres,"*Master's Thesis, NJIT* 1986

[**16**] Govindraju," Computer Simulation for studying the performence of plasticating extruders," *Master's Thesis, NJIT* 1984

Appendix A

CODE FOR PROGRAM

```
/*********************************************************************/
/*          Supprot program to convert assembly language             */
/*          program output to decimal mode for data acquisition      */
/*          purposes                                                 */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

# include <stdio.h>
# include <string.h>
# include <fcntl.h>

typedef  struct data {
                        float points[17];
               struct data  *next;} data1;
data1 *head,*p,*q;
FILE *fpt1;

char header[16][20] = {
                  { 'R' , '.' ,'P', '.', 'M'},
                  { 'R', 'a','t', 'e', '1', 'a'},
                  { 'P', 'r', 's', '1', 'a'},
                  { 'T', 'e', 'm', 'p', '1', 'a'},
                  { 'R', 'a','t', 'e', '1', 'b'},
                  { 'P', 'r', 's', '1', 'b'},
                  { 'T', 'e', 'm', 'p', '1', 'b'},
                  { 'R', 'a','t', 'e', '2'},
                  { 'P', 'r', 's', '2'},
                  { 'T', 'e', 'm', 'p', '2'},
                  { 'R', 'a','t', 'e', '3', 'a'},
                  { 'P', 'r', 's', '3', 'a'},
                  { 'T', 'e', 'm', 'p', '3', 'a'},
                  { 'R', 'a','t', 'e', '3', 'b'},
                  { 'P', 'r', 's', '3', 'b'},
                  { 'T', 'e', 'm', 'p', '3', 'b'},
                  };
int i;
int k;
/*-------------------------------------------------------------------*/

re_ad()     /* read data from the file */
{
     data1 *p1, *q1;
        FILE *fpt1;

fpt1 = fopen ("in","r");

head = (data1 *)malloc(sizeof(data1));

for (i = 0; i <= 15; i++)
fscanf(fpt1,"%f",&head->points[i]);

  head->next = NULL;
```

```
  q1= head;
while (!feof(fpt1))                    /* allocate memory dynamically */
  {
    p1 = (data1 *)malloc(sizeof(data1));
    for (i = 0; i <= 15; i++)
    fscanf(fpt1,"%f",&p1->points[i]);

   p1->next = NULL;
   q1->next = p1;
   q1= p1;
   }
fclose(fpt1);
}
/*-------------------------------------------------------------------------
-----*/
 void pr_int()        /* print the data */
{
 data1 *ptr1;
 data1 *first;
 data1 *temp;
 FILE *fpt;
  ptr1 = (data1 *)malloc(sizeof(data1));
  first = (data1 *)malloc(sizeof(data1));
  temp = (data1 *)malloc(sizeof(data1));
  first = ptr1 = head;

   fpt = fopen("out2.data","w");
while (ptr1->next != NULL)
{
 for (i = 0; i <= 15; i++)
 {
  fprintf(fpt,"\n%-10s",header[i]);
  for(k =0; k <= 3; k++)
  {
   fprintf(fpt," %7.2f",ptr1->points[i]);
   ptr1=ptr1->next;
  }
  temp = ptr1;
  ptr1 = first;
 }
  fprintf(fpt,"\n\n\n\n\n\n\n\n\n\n\n");

 first = temp;
 ptr1 = first;
}
}

/*-------------------------------------------------------------------*/


/*void  co_nvert()                    converts integer to ASCII
int  atoi ()
     {
         int i,n;
```

```
        n = 0;
       for (i = 0 ; s[i] >= '0' && s [i] <= '9' ; ++i)
         n = 10 * n + (s[i] - '0');
       return n;
      } */
/*-------------------------------------------------------------------*/
err_msg()
    {
      printf("INVALID KEY\n");
      printf("Please try again...\n");
      sleep(1);
      men_u2();
     }
 /*-------------------------------------------------------------------*/

conver_t()
  {  int c;
     int  m;
     FILE *fpt3,*fpt4,*fpt5;

  fpt5 = fopen ("test2","w");          /* writes ASCII charecters to a
file*/

  fpt3 = fopen ("data","r");          /* input file pointer */

  fpt4 = fopen ("test1","w");        /* output file ponter */

  while (!feof(fpt3))

    {
       c = getc(fpt3);        /* read from input file */
       putc((c),fpt4);       /* prints to output file */
       fprintf (fpt5,"%d\n",c);     /* converts to ascii */
    }

      fclose(fpt3);
      fclose(fpt4);
      fclose(fpt5);
  }

 /*-------------------------------------------------------------------*/
men_u()
{
    char ch,chois;

    printf("_____________________________________________\n");

    printf (" Please Enter the Choice\n " );

    printf (" 1  -    Start Data Acquisition\n ");

    printf (" 2  -    Quit                    \n ");

    printf ("-------------------------------------------\n");
```

```
    scanf("%c",&chois);

    ch=getchar();

    switch (chois)
           {
               case '1' : printf("Please wait ....\n");
                 sleep(2);
                          men_u2();
                          break;
               case '2' :
                          printf ("BYE BYE ..\n ");
                          break;
                default : err_msg();
                          break;


             }
   }
/*-------------------------------------------------------------------*/
  men_u2()
   {

     char  ch, optio_n;


        printf("-------------------------------------------------\n");

        printf(" 1 - Display data file (Ascii Charecters) \n" );

        printf(" 2 - Convert Ascii Charecters and store   \n" );

        printf(" 3 - Convert data to Ascii file           \n" );

        printf(" 4 - Go Back to Main Menu                 \n");

        printf("-------------------------------------------------\n");

    scanf ("%c",&optio_n);
    ch = getchar();
    switch(optio_n)
         {
            case '1' : optio_n1();
                       break;
            case '2' : optio_n2();
                       break;
            case '3' : optio_n3();
                       break;
            case '4' : optio_n4();
                       break;
            default: err_msg();
                       break;
```

```
        }
    }
/*-------------------------------------------------------------------*/

optio_n1()
{
  char ch,answer;

printf("\n The file for demonstration purpose is TEST.DATA \n");

printf("\n Do You want to see Data in Test.data (y/n) ? \n");

scanf("%c",&answer);
ch = getchar();
         switch(answer){
                        case 'y' : optio_n11();
                                   break;
                        case 'n' : optio_n12();
                                   break;
                        default : err_msg();
                                  break;
                      }
}
/*-------------------------------------------------------------------*/
optio_n11()
{
     FILE *fpt6;
     int c,t;
     char ch;

     fpt6 = fopen("test.data","r");

   while (!feof(fpt6))
    {
      c = getc(fpt6);
     putc(c,stdout);
     }
  ch = getchar();
  men_u2();
 }
/*-------------------------------------------------------------------*/
optio_n12()
     {
       FILE *fpt7;
       char ch,file_name[16];
       int c,t;


        printf("\n Please Enter The File name\n");

        printf("\n      Warning!                \n");

        printf("\n For Data Acquisition purpose the file \n");
```

```
        printf("\n should be in the ASCII format \n");

        scanf("%s",file_name);

        fpt7 = fopen(file_name,"r");
        while (!feof(fpt7))
          {
             if (fpt7 = NULL )
            {
             printf("\n Can not find the file ..\n");
             printf("\n Please Try again....\n");
             men_u2();
             }
            else
            {
              c = getc(fpt7);
             putc(c,stdout);
            }
          }
       ch = getchar();
       men_u2();
      }

/*-----------------------------------------------------------------*/
optio_n2()
  {
     FILE *fpt8,*fpt9;
     char ch,file_name[];
     int c,t;

    printf("Please Enter  the file name\n");

    printf("\n File for demo. purpose is test.data\n");

    printf("\n The Output will be available in  out.data\n");

    scanf("%s",file_name);

    fpt8 = fopen(file_name,"r");
    fpt9 = fopen("out.data","a");
    while (!feof(fpt8))
     {
        fscanf(fpt8,"%d\n",&c);
        putc(c,fpt9);
        putc(c,stdout);
     }
     fclose(fpt8);
     fclose(fpt9);
   ch =getchar();

  men_u2();
 }
/*-----------------------------------------------------------------*/
```

```
optio_n3()
{
  FILE *fpt10,*fpt11;
 char  ch,file_name[16] ;
 int c,t;

  fpt11 = fopen("asc_file","w");
  printf("\n Please Enter the file to be converted \n");
  scanf("%s",file_name);
 printf(" The output will be avialable in file  asc_file\n");
 fpt10 = fopen(file_name,"r");
  while (!feof(fpt10))
    {
       c =  getc(fpt10);
       fprintf(fpt11,"%d\n",c);
       fprintf(stdout,"%d\n",c);
     }
  fclose(fpt10);
  fclose(fpt11);
 ch = getchar();
 men_u2();
 }
/*-------------------------------------------------------------------*/
optio_n4()
{
 men_u();
}

 /*-------------------------------------------------------------------*/
main()
       {                              /* main program */
        re_ad();
        pr_int();
        men_u();
        }
```

```
;-------------------------------------------------------------
;Assembly language program to load the file and
; execute it.
;
;
;Module -1
;
;-------------------------------------------------------------

prefix  segment at 0          ; program prefix  segment template
        org     2ch
envseg  dw      ?
prefix  ends
; parameter block. Pointed by ES:BX before a program is loaded
dseg      segment  public 'data'
pblock    dw       ?          ;segment address of environment
          dw       80h        ;
          dw       ?          ;Segment pointer to command line
          dw       5ch        ;
          dw       ?          ; segment pointer to default FCB
          dw       6ch        ;
          dw       ?          ;segment pointer to 2nd default FCB
spsave    dw       ?          ;storage area for SS:SP
sssave    dw       ?          '
filname   dw       'asc_file',0
errmsg    db       'File not found',0dh,0ah,'$'
msg1      db       'program starting',0dh,0ah,'$'
msg2      db       'Program completed',0dh,0ah,'$'
dseg      ends

sseg      segment stack 'data'
          dw      80h     dup (?)
sseg      ends

cseg      segment public 'code'
          assume    cs:cseg, ds:dseg
setup     proc      far
          cld                     ;all string moves go up
          push     ds
          sub      ax,ax
          push     ax
          push     ds            ;print a program starting message
          mov      ax,dseg
          mov      ds,ax
          lea      dx,msg1
          mov      ah,9
          int      21h
          pop      ds
          assume   ds:prefix
          lea      bx,eop+10h       ; get paragraph # of end of
prog.
          mov      cl,4
          shr      bx,cl
          add      bx,cseg
```

```
          mov     ax,ds             ;get paragraph # of program
start
          sub     bx,ax             ;difference is total prog.
length
          mov     es,ax             ;Point ES at program prefix
segment
          mov     ah,4ah            ;Shrink allocated memory to
prog size
          int     21h
          mov     bx,envseg         ;Get segment address of
environment
          mov     dx,ds             ;get program prefix segment
address
          mov     ax,dseg           ;print DS to local data area
          mov     ds,ax             ;
          assume  ds:dseg
          lea     di,pblock         ;fill in parameter block
          mov     [di],bx           ;with environment segment
          add     di,4
          mov     cx,3
filblk:   mov     [di],dx          ;AND command line and FCB
pointers
          add     di,4
          loop    filblk
          lea     bx,pblock        ;point   ES:BX at parameter
block;
          mov     ax,ds
          mov     es,ax
          lea     dx,filnam         ;point DS:DX
          mov     spsave,sp         ;save stack location
          mov     sssave,ss         ;
          mov     ax,4b00h          ;load and execute new program
          int     21h
          mov     ax,dseg           ;restore ds to local area
data
          mov     ds,ax
          mov     ss,sssave         ;then restore original stack
          mov     sp,spsave
          jnc     finish             ;leave if CF not set on
return
          lea     dx,errmsg         ;if CF set, print an error
message
          mov     ah,9
          int     21h
          ret
finish:   lea     dx,msg2        ;print a "program completed"
message
          mov     ah,9
          int     21h
          ret                          ;and quit
setup     endp
eop       db      'end of program'

          end     setup
```

```
;-------------------------------------------------------------
;Assembly language program to write the data to
;the file in  ASCII format.
;
;
; Module 2
;-------------------------------------------------------------
;
dseg      segment
bufr      db          'This is a disk file'
filnam    db          'anyfile.txt',00
crerr     db          'Error : cannot execute
file',0DH,0AH,"$"
dseg      ends


sseg      segment   stack     ; The dos will automatically
setup
          dw        80h  dup(?); a stack in the stack
segment
sseg      ends


cseg      segment
          assume cs:cseg,ds:dseg,ss:sseg

main      proc    far
          push    ds        ;push atart address at prog.
prefix
          sub     ax,ax     ; segment on the stack
          push    ax

          mov     ax,dseg    ;point DS:DX at file name in

dseg
          lea     dx,filnam
          mov     cx,0       ;make file attribute byte 0
          mov     ah,3ch     ;creat the file
          int     21h
          jc      error       ; abort if there was a problem
          mov     bx,ax       ;else store file in BX
          mov     cx,36      ;
          lea     dx,bufr    ;
          mov     ah,40h     ; close the file
          int     21h        ;
done :    ret                ; far return gets back to dos


error:   lea      dx,crerr    ; if error occures during file
creation
         mov      ah,9        ; print an error message
```

```
        int     21h         ;
        ret

main    endp
cseg    ends
        end       main
```