A proposal for a development platform for microcontroller-based devices

Michael L. Wetton

Edith Cowan University

A PROPOSAL FOR A DEVELOPMENT PLATFORM FOR MICROCONTROLLER-BASED DEVICES

By

Michael Leon Wetton BSc.

A Thesis Submitted in Partial Fulfilment
of the Requirement for the Award of

Master of Science

at the School of Mathematics, Information Technology and Engineering.

Perth
Western Australia.

Date of Submission: 11th June, 1995

ABSTRACT

This thesis is concerned with designing, implementing and testing a miniaturised temperature data logging device. Investigations demonstrated that a microcontroller could provide a low-cost single-chip solution to this problem and after a detailed review of 8-bit microcontrollers, the MC68HC11 was chosen for this task. This document also includes discussion on an environment that was developed for creating and testing MC68HC11 software and the use of Motorola's evaluation boards. To ensure that the device was designed to software engineering standards an investigation into software engineering analysis techniques took place. This resulted in the Jackson Structured Programming (JSP) methodology being adapted to produce a proposed development platform suitable for microcontroller-based design.

DECLARATION

I certify that this theses does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any institution of higher education and that, to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where due reference is made in the text.

Michael Leon Wetton

11th June, 1995

ACKNOWLEDGMENTS

I thank Edith Cowan University for providing the means to complete this study and for providing several Higher Education Contribution Scheme Exemption Scholarships.

I thank Dr Paul Maj and Dr Thomas O'Neill, my supervisors, for their advice, guidance and support during the years of my study.

I thank Mrs Jennifer Renner and other members of the library staff who provided assistance during my library searches.

I also thank Wendy, my wife, for the moral support during the years of my study.

Table of Contents

Volume 1

Volume 2

APPENDICES

List of Figures

List of Tables

REFERENCES

REFERENCES

Alvey (1986) *The Derivation of Standards for programming Practices and the Tools to enforce them,* UK Alvey Programme, Project Reference SE/058, London.

Aschoff J (1982) *Circadian Rhythms in Man,* "Biological Timekeeping", Society For Experimental Biology, Seminar Series, Cambridge University Press, Cambridge.

Bell D, I. Morley and J. Pugh (1987) *Software Engineering: A Programming Approach,* Englewood Cliffs, N. J., Prentice/Hall, London.

Booch G (1986) *Object-Oriented Development,* IEEE Trans. Software Engineering , vol SE-12, no. 2.

Brady J (1982) *Biological Timekeeping,* Cambridge University Press, Cambridge.

Brown, Christmas and Ford (1992) N-Z-Med-J

Briggs A (1991) *Circadian Rhythms: Temperature and Cognitive Functioning of the Institutionalized Elderly,* Degree: MS, California State University, Fresno.

Cameron J (1983) *The Jackson approach to Software Development,* IEEE Computer Society Press, Washington.

Coad P, E. yourdon (1990) *Object-Oriented Analysis,* Prentice/Hall, London

Cooling J (1991) *Software Design for Real-Time Systems,*

Davies C (1987) *An Investigation into Computer Assisted Program and System Design,* The British Library Document Supply Centre, West Yorkshire, UK.

DeMarco T (1979) *Structured Analysis and System Specifications,* Prentice/Hall, London.

Edwards H (1990) *Interfacing and Programming Methods,* The British Library Document Supply Centre, West Yorkshire, UK.

Floyd T (1984) *Electronic Devices (second edition),* Merrill Publishing Company, London.

Fraden and Lackey (1991) Clin-Pediatr-Phila

Geitner M (1991) *An Investigation into the Relationship Between Circadian Rhythm Perceptions and Lifelong Learning,* Degree: EDD, Northern Illinois University.

Hall A (1990) *Seven Myths of Formal Methods,* IEEE Software (Septermber).

Hashimoto and Okamoto (1990) *A Set and Mapping-based Detection and Solution Method for Structure Clash between Program Input Output Data,* ATR Communication Research Laboritories, Kyoto 619-02, Japan.

Hoare C (1985) *Communicating Sequential Processes,* Prentice/Hall International, London.

Horowitz P and W. Hill (1980) *The Art of Electronics,* Cambridge University Press, Cambridge.

Hu D.(1990) *Object-Oriented Environment in C++,* Advanced Computer Books, MIS Press, Delran, N. J.

Jackson M (1975) *Principles of Program Design,* Academic Press,

Jackson M (1983) *System Development,* Prentice/Hall, Englewood Cliffs, N. J.

Jacob J (1989) *Industrial Control Electronics applications and design,* Prentice/Hall of Australia Ltd, Sydney.

Littlewood B (1987) *Software Reliability: Achievement and Assessment* Blackwell Scientific Publications, Oxford

Mano M (1988) *Computer Engineering Hardware Design,* Prentice/Hall International Editions Inc, London.

Meyer B (1988) *Object-Oriented Software Construction,* Prentice/Hall, London.

Milewski, Ferguson and Turndrup (1991) Clin-Pediatr-Phila

Mohri and Kikuno (1991) *Fault Analysis Based on Fault Reporting in JSP Software Development,* System Education Human Resources Development Department, Nihon Unisys Ltd, Kanagawa, Japan.

Morley, Hewson,Thornlton and Cole (1992) Arch-Dis-Child

Naylor E (1985) *Tidal and lunar rhythms in animals and plants,* Biological Timekeeping, Society For Experimental Biology, Seminar Series, Cambridge University Press, Cambridge.

Newson T (1991) *Sick Leave Patterns of Nurses on Permanent and Rotating Shifts,* Texas Woman's University.

Noble J (1992) Pediatr-Emerg-Care

Orr K (1977) *Structured Systems Development,* Yourdon Press, New York.

Ostesen M (1991) Ugeskr-Laeger

Pressman R (1992) *Software Engineering: A Practitioner's Approach (third edition,* McGraw-Hill, New York.

Ringer C (1972) Medical Thesis

Ritter D (1988) *And We Were Tired: Fatigue and Aircrew Errors,* NASA Ames Research Centre.

Robertson. Bradkey and Fullen (1991) Clin-Pediatr-Phila

Roper R (1988) *The Derivation of a Methodology with Supporting Software Aids for Testing Structured Data Processing Programs,* The British Library Document Supply Centre, West Yorkshire, UK.

Roper R and Smith (1989?) A formal Program Methodology For Use In The Development Of Software"

Rozman (1989?) *Software Engineering: How to Approach it to the Electrical Engineer,* University of Naribor, Naribor, Yugoslavia.

Saunder D (1986) *Computers Today (second edition)* McGraw-Hill Book Company, Ney York

Shenep, Adair,Hughes,Flynn

Shlaer S and S. Mellor (1988) *Object-Oriented Systems Analysis*, Yourdon Press, London.

Sommerville I (1987) *Software Engineering,*

Stevens W G. Meyer and L. Constantine (1974) *Structured Design,* IBM Systems Journal, vol 13, no. 2.

Still H (1972) ,*Of Times, Tides,and Inner Clocks,* Stackpole Books,Harrisburg.

Tocci R and L. Laskowski (1986) *Microprocessors and Microcomputers (M6800),* Prentice/Hall, Englewood Cliffs, New Jersey.

Thompkins W and J. Webster (1988) *Interfacing Sensors to the IBM PC* , Prentice/Hall, Englewood Cliffs, New Jersey.

Warnier J (1981) *Logical Construction of Systems,* Van Nostrand Reinhold.

Wiener R and Sincovei (1984) *Software Engineering with Modular 2 and Ada,* Wiley.

Wing J (1990) *A Specifier's Introduction to Formal Methods,* IEEE Computer, vol 23, no. 9, September.

Wobschall D (1987) *Circuit Design for Electronic Instrumentation,*

Woodcock J (1989) *Calculating Properties of Z Specifications*, ACM Software Engineering Notes, vol 14, no. 5.

Yang S (1990) *Computer Modelling and Analysis of Biological Rhythms*

Yourdon E and L. Constantine (1979) *Structured Design, Fundamentals of a Dicipline of a Computer Program and System Design,*

Yourdon E (1989) *Modern Systems Analysis,*Prentice/Hall.

Electronic Data Sheets and Micrprocessor Manuals

****

Texas Instruments (1989) Data Book volume 1 (TTL)

Texas Instruments (1989) Data Book volume 2 (TTL)

RS data sheet 3992 (1983) Semiconductor temperature sensor (RS590)

RS data sheet 1867 (1983) Thermistors

RS data sheet 8307 (1987) Temperature sensor IC (LM35)

RS data sheet 3374 (1983) 16-key encoder

RS data sheet 5314 (1984) Trimmable voltage reference IC (ZNREF040)

RS data sheet 9811 (1989) Alphanumeric Dot matrix LCD with backlighting

RS data sheet 6569 (1986) Alphanumeric dot matrix LCD

Seiko Instruments Inc (1988) LCD module L4042 user manual (an.no.L4042-840E

Motorola (1989) M68HC11EVB evaluation board (BR278/D)

Motorola (1989) M68HC11EVM evaluation board (BR266/D)

Motorola (1988) HCMOS Single-Chip Microcontroller

Motorola (1991) HC11 M68HC11 reference manual

Motorola (1988) Real-Time Clock plus RAM (RTC) MC146818

Motorola (1988) RTC plus RAM with serial interface MC68HC68TI

Motorola (1990) Universal Evaluation Board user's manual EVBU

Motorola (1986) Evaluation Board User's Manual EVB

Motorola (1989) Evaluation Module User's Manual EVM

Motorola (1986) 8-bit HCMOS Microcomputer MC68HC811A2

Motorola (1991) 8-bit Microcontroller MC68HC711J6

Motorola (1991) 8-bit Microcontroller MC68HC11L6

Motorola (1990) 8-bit Microcontroller Unit MC68HC711K4

Motorola (1991) 8-bit Microcontroller MC68HC11D3 / D0

Motorola (1991) 16-bit Modular Microcontroller MC68HC16Z1

Motorola (1991) 8-bit Microcontroller Unit MC68HC705H2

Motorola (1988) MC68HC11 EEPROM Programming from a PC

National Semiconductor (1989) The 8-Bit COP800 Family

National Semiconductor (1989) The 8-Bit COP888CL Single-Chip Microcontroller

NEC (1992) Single-Chip Solution with 4/8/16-bit Microcontrollers

Intel (1988) 8-Bit Microcontroller 8051

Intel (1988) 16-Bit Microcontroller 8098

CHAPTER ONE

1. INTRODUCTION

The overall aim of this thesis is to propose a development platform that can be used to design microcontroller-based devices. The steps taken to design the hardware of a small monitoring device, and its support system are described in detail. Then the selected Jackson Structured Programming methodology describes how software, used to control a small monitoring device and support system, can be designed, implemented and tested.

The device is required to measure, and store, a person's body temperature at regular intervals over a lengthy period of time (typically, four weeks). Furthermore, its size is to be small enough so that it may be worn by a person in a manner not likely to cause discomfort or inconvenience. At the same time, the device's support system used for data processing is fashioned to be practical, simple and easy to use.

It is common knowledge that the human body is affected by such criteria as:

- a long flight on a jet aircraft over several time zones, which causes jet lag, or
- shift work, which produces listlessness due to a change in environmental conditions.

In such cases, the human body undergoes a time `warp' giving rise to a problem in which the body's active and passive daily phases have been significantly influenced. Alternatively, the body's circadian rhythms, which are daily biological processes dependent upon internal clocks, suffer a severe phase shift.

According to Aschoff (1982) these internal clocks affect alertness, speed of reaction and speed of computation; and, he states that there is a correlation between these three effects and body temperature. Consequently, a body temperature monitoring device should prove invaluable to people conducting research into circadian rhythms and their effects on body functions.

During the late 1980's and early 1990's several multi function processing devices (such as microcontrollers) were produced by corporations like Motorola, Intel, NEC and National Semiconductor. A study of these microcontrollers is included as they enable a single-chip solution to the problem of miniaturising data logging devices. Considerable emphasis has been placed on the use of software engineering in the design, implementation, validation and documentation of such systems. Such practices lead to an increased confidence in the reliability of a design and helps to ensure a device that achieves a solution to the problem at hand. Furthermore, an outline is given of the development platform used for designing systems that utilise microcontrollers. The temperature monitoring system developed is an example of a single-chip solution to the problem of miniaturisation.

Chapter two contains a review of background material needed for device, and system, design in the field of miniaturising data logging systems for measurement of human body temperature. More specifically, we discuss at length the following topics:

- human circadian rhythms and their influence on human body functions
- measurement of human body temperature via intrusive and non-intrusive methods
- identification of criteria to aid the selection process of an appropriate body temperature sensor

- comparisons of various microcontrollers and their features to enable selection of the most suitable one for our system's needs
- creation of a platform to facilitate the development, implementation and testing of software and microcontroller support circuitry

The discussion shows that there is a need for miniaturised data logging devices, as a cost effective means of corporal data acquisition.

Chapter three describes a hardware design of a system that measures, records, and shows graphically, human body temperatures.

Firstly, the design of two temperature sensor circuits are described in detail. One scheme incorporates a LM35 integrated circuit sensor, the other uses a YS44002 thermistor as a sensor. Both circuits are powered by a +5 volt supply and include a voltage reference zener diode to enable temperatures to be monitored with a +/- 0.1 degree Celsius accuracy. Then, the functions required to implement :

- a temperature monitoring device,
- a data transfer device, and
- a data processing system

are listed and analysed. Finally, the results of the hardware design are given in diagrammatic form in figures 7 through to 13.

Chapter four describes the temperature monitoring system from a Software Engineer's point of view. It contains a description of the System Model in which the relationship between the hardware and the user is discussed.

The software requirements documentation has items such as:

- hardware specifications, which describe the requirements from a users point of view,
- functional requirement specifications, that include: all the inputs, outputs, expected error situations, solutions to the expected errors and the processes required to be performed,
- data type requirements, include: microcontroller I/O registers, program parameters, variables, memory buffers and initial values used when the system is reset,
- non functional requirements, which specify how well a function should be performed, how the system connects to its environment, the limits placed on the design and any other constraints given to the system, and
- maintenance and testing information, in particular, details of a test plan which incorporates: functional testing, module testing, system testing and acceptance testing of the overall system.

The design of software employs a Jackson Structured Programming (JSP) methodology, which includes:

- a structured analysis technique to create data flow diagrams,
- transform analysis techniques to convert data flow diagrams into Jackson structure diagrams and
- Jackson Structure Diagrams, which categorises the logic into three types of processes: namely, sequence, choice and iteration.

A diagrammatic methodology was chosen in order to show the flow of a program's structure in an easy to follow manner.

Chapter five is reserved for concluding remarks. It provides a statement of the original contributions of this thesis and some thought towards future developments and research.

Finally, the appendices, contains three sets of: data flow diagrams, Jackson structure diagrams and the associated program listing for:

- the temperature monitoring device,
- the data transfer device, and
- the data processing system.

CHAPTER TWO

2 BACKGROUND MATERIAL

2.1. CIRCADIAN RHYTHMS IN HUMANS

2.1.1. INTRODUCTION

This section of the notes is written to illustrate the importance of measuring body temperature, in the field of medicine. It is especially important when studying the effects of stress in relationship to humans' circadian rhythms.

The Collins dictionary definition for **circadian** is, "an adjective which describes the biological processes that occur regularly at 24 hour intervals". The Latin meaning for circadian is, "about a day".

It is common knowledge that both animals and plants behave differently depending upon whether it is day or night. They both have internal clocks which, under normal conditions, synchronise with the light-dark cycle. The internal clocks may adapt to other criteria; for example, the seasons, temperature cycles and even social issues. Naylor(1982) states that "animals have various types of internal clocks. some are affected by internal conditions, for example, one of them, the heart, beats on demand". The other set of clocks are affected by external conditions; they have environmentally related rhythms as follows:

the 24 hour day,
the 12.4 hour high tide,
the 14.8 day spring tides,
the 29.5 day lunar month, and
the 365 day year.

Circadian rhythms generate patterns of locomotor activity alternating with rest or sleep. Some people believe that the reason for sleep is to enforce inactivity in animals to reduce the risk from predators. A more traditional view is, that sleep restores body reserves.

2.1.2. MAN AND THE CIRCADIAN RHYTHMS

Man adjusts to the environment; his 24 hour internal clock prepares him for efficient activity during the day and rest at night. Many of man's structures and functions undergo regular 24-hour changes. The human circadian system consists of multiple biological oscillators which are normally coupled to each other giving rise to a stable internal clock.

The effects of the internal clock can be seen by analysing the variety of rhythms that can easily be measured under experimental conditions. They are:

sleep-wakefulness,
alertness,
speed of reaction,
speed of computation, and
body temperature.

2.1.3. ANALYSING CIRCADIAN RHYTHMS BY MEANS OF TEMPERATURE

It should be noted at this point, that most of the rhythms that can be demonstrated in man, have a similar wave shape. Under normal conditions, they all have their peak during the daytime, and their low during the night. (see Figure 1)

It could be said that, when measuring the body temperature of a person, the result gives a fair indication of the potential useful activity in the other areas of interest. For example there is a correlation between body temperature and sleep. There is also a correlation between body temperature and speed of reaction, speed of computation and alertness.

The variation of temperature ranges from approximately 37.5 degrees Centigrade at the daytime peak, to approximately 36.0 degrees Centigrade as a low at night. Thus, any experiment that is designed to monitor circadian rhythms with respect to temperature, would need a temperature measuring device capable of measuring a range from 30 degree Centigrade to 40 degrees Centigrade, with a steps of 0.1 degrees (+/- 0.1 `C error). Note, relative temperature changes of a body are more important than the actual temperate values themselves. As a matter of interest, the maximum temperature is normally detected in the late afternoon and, the minimum temperature would normally be detected during the second half of a sleep pattern.

From an engineering point of view, measurement of the required temperatures could be achieved with an Integrated Circuit (I.C.) temperature sensor, an Analog to Digital Converter (A.D.C.) and a means of storing the results.

2.1.4. DESYNCHRONISATION OF CIRCADIAN RHYTHMS

There are two main sets of oscillators associated with the human body: one set controls wakefulness and sleep (this is highly variable in frequency), and the other set controls the temperature rhythm which is relatively stable. In abnormal situations , such as sleeplessness, the temperature and activity no longer correlate closely.

Jet Lag

Aschoff(1982) states that "modern air travel gives everyone the opportunity to see how the circadian system can be upset". A long flight across several time zones has the following effect:

(i) At first the circadian rhythm is unaffected, but out of synchronisation with the local time.

(ii) It takes several days to regain a normal phase relationship with the new environment.

(iii) People make errors of judgment during the first three days after the time zone change.

(iv) It takes five days to have a clear rhythm again.

(v) It takes eight days to have a normal rhythm in synchronism with the new environment.

Shift Work

Aschoff (1982) also states that a strong contrast to jet lag is the situation in which the shift worker has to suffer. Shift workers have a confusing environment. They have to react to an artificial light-dark cycle on one hand, whereas, the other environmental time signal, like family activity, are phase shifted. The low temperature readings during sleep may shift from early to late sleep, over a period of about 20 days. The high temperature reading also gradually moves from early in the work period to a later time, over a period of about 20 days.

2.1.5. RESEARCH RELATING TO CIRCADIAN RHYTHMS

Research is still being carried out in order to relate the effects of circadian rhythms to people's learning capacity and work performance. This section of the thesis shows that there is still a real need to monitor people's temperature and corresponding activities.

A description correlational study was conducted to explore the relationship between body temperature and the process of acquiring knowledge by institutionalised elderly people. Briggs (1991) .

An investigation was conducted into the relationship between circadian rhythm perceptions and learning as determined by academic achievement.
Geitner (1991).

Yang (1990) states that "biological rhythms are an important phenomenon and feature of physiological systems. Indirect means have to be employed for their description and exploration due to the unclear internal nature of the systems. Research was carried out on the frequency correlation between two different circadian rhythms: temperature and activity".

A retrospective study was conducted by Newson (1990) to determine if there was a difference in use of sick leave by nurses working:

(a) permanent night shifts,
(b) permanent evening shifts and
(c) rotating shifts.

Ritter (1988) research was aimed at demonstrating that the majority of errors made by aircrew members are cognitive errors, not control errors, and that a major contributing factor was fatigue. He argued that fatigue is increased by sleep deprivation, circadian desynchronosis, and poor scheduling practices.

2.1.6. THE MEDICAL ASPECTS

The importance of the circadian system to the medical field is due to the following:

(a) a high degree of temporal order relates to a healthy body,

(b) the response time of a body to react to a stimulus (drugs) depends upon the circadian phase, and

(c) there are drastic changes from hour to hour that occur in many of the circadian variables that are measured for diagnostic purposes.

2.1.7. CONCLUSION

It seems obvious that any means of measuring low resolution temperatures (0.1 degrees Celsius) without any stress or uncomfortable feeling to a patient, or any person taking part in an experiment, is a desirable tool to the medical profession. The smaller and lighter the device the better the tool would be as it needs to be worn continually for extensive periods.

2.2. MEASUREMENT OF HUMAN BODY TEMPERATURE

2.2.1. INTRODUCTION

Normally an adult's body temperature remains constant for a particular time of the day. There is a circadian rhythm of body temperature, in which the body temperature reaches a peak during the wakeful day and a low during sleep or restful night. The difference between the normal circadian maximum and minimum body temperatures is approximately 1.5 degrees Celsius.

Any variation from the normal body temperature indicates that there is possibly something wrong with the functions of the body.

The average normal body temperature is said to be 37 degrees Celsius. If the body temperature rises significantly above 37 degrees Celsius , then the person is said to have a fever; an abnormally high body temperature.

The following pages are designed to show that it is important to know <u>when, where and how</u> to measure body temperature.

2.2.2. THE BODY TEMPERATURE MEASUREMENT SITES

Doctors, nurses and surgeons are interested in a patients core temperature to establish the state of their health. The core temperature is defined as the temperature of the blood passing through the pulmonary artery.

Traditionally, doctors and nurses use temperatures taken from the following three body sites: **oral, axillary** and **rectum**. Temperatures from these main body sites have been used to predict the human body's core temperature.

However, in recent years (1991 and 1992) several researchers including:

Noble(Feb 92),

Fraden and Lackey(April 91),

Shenep, Adair, Hughes, Robertson, Flynn, Bradkey and Fullen (April 91)

Milewski, Ferguson and Turndrup (April 91)

have published articles recommending the use of a new method of measuring body temperature from a fourth body site (the ear). The new method uses an infrared ear thermometer (also called the tympanic membrane thermometer). This device measures temperatures from within a person's ear canal. It gives a reliable, non invasive, quick method of measuring body temperature.

Hence, there are now four body sites that may be used to easily measure body temperatures:

- oral,
- axillary,
- rectum,
- ear.

2.2.3. DIFFERENCES IN TEMPERATURE MEASUREMENT

Firstly, it is important to know that the body temperature of a normal healthy person has a circadian rhythm. Although the figure of 37 degrees Celsius is said to be the temperature of a normal healthy adult body, it can easily be shown that the body temperature of humans may reach a peak of 37.5 degrees Celsius during the wakefulness of day and be at a low of 36.0 degrees Celsius during a restful, or a sleep filled night. (see Figure 1)

Secondly, the temperature taken from the four main body sites: oral, axillary, rectum and the ear, varies from the required core temperature:

- oral gives the most accurate reading,

- axillary gives a reading that is 0.5 degrees Celsius lower than the core temperature,

- rectum gives a temperature that is 0.5 degrees Celsius higher than the core temperature.

- the reading from the ear with a tympanic membrane thermometer depends upon the ambient temperature, which means that a small calculation is required to accurately predict the core temperature.

If the variants are taken into account, then all four methods are acceptable ways for predicting the core temperature of the human body.

2.2.4. THE EFFECT OF A PERSONS AGE

Empirical data shows that infants under the age of 12 months, and aged people over the age of 80 years, have greater variations in body temperatures than adolescent and adult people. For example, studies by Brown, Christmas and Ford (1992) have shown that, "The current clinical practice in assessing infant body temperature by using axillary temperatures does not reflect rectal temperatures in a reliable constant fashion".
Hence, infants and aged people need to be considered as special cases when designing instruments for measuring body temperature.

2.2.5. THE DURATION OF TEMPERATURE MEASUREMENTS

Over the years many studies have looked into the amount of time that a thermometer has to be present at the body site, to ensure that they have fully acquired the body temperature.

Usually nurses wait at least one and a half minutes before removing a mercury thermometer from a patients body site.

Ostesen (1991) conducted a study that investigated, "Whether the rectal Craftemp measurement of temperature could be used as an alternative to the measurements with a mercury thermometer", and came up with the following discovery. That the one minute time of measurement recommended by the manufacturer of the electronic temperature measuring device (called Craftemp) was insufficient. It should have been 2 minutes.
Note, the time measurement problem does not arise with sensors attached permanently to a body site. Thus, a small comfortable temperature measuring device that is permanently attached to the body, and is continually monitoring body temperature would not suffer from the time measurement problem.

Noble (1992) informs us in a journal article that the latest infra-red (IR) ear thermometer which allows users to take a quick and non invasive measurement of body temperature is also desirable device.

2.2.6. THE INSTRUMENTS WHICH MEASURE BODY TEMPERATURE:

There are various ways of measuring body temperature. This section subdivides temperature measuring devices into 4 categories:

- **(i) glass - mercury thermometers,**
- **(ii) electronic instruments,**
  - **(a) thermistors**
  - **(b) I.C. temperature sensors**
- **(iii) infrared ear thermometers**
- **(iv) other means of measuring body temperature,**
  - **(i) magnetic resonance**
  - **(ii) thermadot disposable**

The following sections describe the sensors from the first two sections shown above. The glass-mercury thermometer is described because it has been the medical professions standard temperature measuring instrument. The thermistors and IC devices because they offer a small and convenient way of measuring body temperature of an active person under test.

(i) The glass - mercury clinical thermometer,

The clinical thermometer that is made from glass and mercury is specially designed for measuring body temperatures of humans. It has graduation marks that show a range of temperatures between 35.0 degrees Celsius and 43.0 degrees Celsius in 0.1 degree steps.

This instrument requires at least 2 minutes to ensure that it fully acquires the body temperature.

Glass mercury thermometers consist of an envelope of glass that houses a large bulb which contains all the mercury at room temperature. Attached to the bulb is a small capillary tube that allows the mercury, that is expanded by the body heat, to flow. The mercury in the capillary tube cannot return to the bulb of mercury easily because of a restriction in the capillary tube close to the bulb of mercury. Thus, when the thermometer is removed from a patient the mercury remains in the capillary tube and the measurement can be recorded with with reduced error. The instrument has to be allowed to cool, and then shaken to force the mercury in the capillary tube back into the bulb.
The advantages of measuring human body temperatures with a glass thermometer are: it has become the accepted standard temperature measuring device and the device can easily be sterilised. The disadvantage is that it cannot be used for automated recording of temperatures.

(ii) electronic instruments:

Nearly every electronic property of a material varies as a function of temperature, and could in principle be employed as a temperature sensor. It is only the requirements of : high sensitivity, reproduciblity, and linearity that limit the possibilities, especially if cost, size, and ease of readout are also considered.

Thermistors and Integrated Circuit (IC) sensors are considered to be the most suitable electronic components for measuring body temperature.

(a) thermistors

Basically a thermistor is a resistor with a high temperature coefficient. It is a semiconductor that is found in various geometrical configurations, to which leads are attached. In fact they are found in a wide variety of shapes and sizes; down to microscopic sizes. They have a negative temperature coefficient; their resistance decreases with increasing temperature. (see Figure 2)
The bad points associated with thermistors are:

> The resistance-temperature variation is non linear; over most of its range the resistance decreases exponentially with increasing temperature. Thermistors are affected by internal heating (power dissipation) caused by the voltage applied to the thermistor from the readout or converter unit.

The good points associated with thermistors are:

> Although thermistors are not highly precise sensors, they are popular for their low cost, high sensitivity, ease of readout and small size.
>
> A convenient configuration can also be found to suit a particular application; for example, a bead for measuring internal body temperatures, a thin disc form for measuring skin temperatures.

(b) IC temperature sensors

The IC temperature sensors are based on the diode voltage being temperature dependant. One version (the AD590) has a current output proportional to absolute temperature.

The sensor is insensitive to the voltage across it and can be used even with long lead wires. Another version (the LM335) has an output voltage proportional to temperature.

The good points associated with IC temperature sensors are:

their output is linearly proportional to temperature,

their time constant is reasonable; 60 seconds in still air, 1.4 second with a heat sink.

maximum error is less than, plus or minus, 0.05 degrees Celsius

The bad points associated with IC temperature sensors are:

Their shape is usually in the form of a transistor package (bulky).

Their output requires amplification

(AD590 = 1 micro amp/degree Kelvin) or

(LM335 = 10 milli volts/degree Kelvin)

2.2.7. HUMAN/INSTRUMENT INTERFACE

The most obvious site for measuring human body temperature, when taking readings from a permanently fixed sensor over a long period of time, is the axillary site. The temperature readings from this site are said to be correlated to the core temperature of a human body, and there should be no inconvenience to the person under test, as long as the sensor is small.

The duration of the temperature measurements should not cause a problem as the sensor would be permanently fixed to the person under test.

DATA STORAGE

Readings would be taken, by the temperature monitoring device, every ten minutes. This would give:

6 * 24	=	144	readings a day
7 * 144	=	1008	readings a week
4 * 1008	=	4032	readings during a 4 week test period

Therefore, the measuring device needs to be able to store at least 4032 temperature readings when used over a four week period. Given that conventional devices measure temperature between a range of 35.0 through to 43.0 degrees Celsius in 0.1 degree steps. This means that about 80 relative temperature values are possible for each reading. Note, that each reading can be stored in one byte of memory if an electronic measuring device is used. One byte has 8 bits of information (2 to the power of 8 different codes) which allows 256 possible values per reading)

Consequently, a 4K byte memory chip (RAM or EEPROM) can be used to store four weeks worth of data.

2.3. CHOOSING A TEMPERATURE SENSOR

2.3.1. SENSOR SELECTION CRITERIA

Choosing a temperature sensor to measure human body temperature requires a set of criteria to be considered.

Firstly, various selection criteria, mentioned in books and manufacturer's data sheets, were listed and then analysed. The criteria selected for consideration included: accuracy, stability, linearity, temperature coefficient, response time, power dissipation constant, ruggedness or fragility, ease of readout, cost of manufacture, resistance to chemical attack, requirement of a reference temperature, self-power character, sensitive to interference, suitability for remote sensing, the required temperature range, self-heating effects, and the choice of shapes and sizes. The significance of each criterion, relating to measurement of human body temperature, had to be realised.

Secondly, various types of temperature sensors were considered. Matching the criteria, important to this project, with the commerially available sensors helped to reduce the selection down to three types of sensors, namely: thermistors (YSI 44000 series), IC current sensors (AD590) and precision IC sensors that produce a voltage output (LM 34 and LM 35).

The third stage of the task was to evaluate circuit designs for the three chosen types of sensors. This involved circuits being designed, built and tested. This enabled preliminary results to be analysed and the complexity of the circuits to be considered.

The design of the sensor support circuitry involved the following:

selecting a suitable precision voltage reference zener diode,

calculating resistor values to minimise the currents flowing through the circuit, and

calculating resistor values that enable a suitable range of voltages to be input into a microcontroller system.

The overall result was that the IC current sensor was eliminated, from the three chosen sensors, because of the size and complexity of the support cicuitry. For example, the circuit required an operational amplifier and more importantly an additional + and - power supply.

Note, the most important criterion of the temperature monitoring system was that the device had to be as small as possible. Hence, the LM 34 IC voltage sensor and the YSI 44002 thermistor, used as a temperature sensor, were the only two devices left for further analysis. Both devices can be shown to produce an accurate voltage output, proportional to temperature changes of their environment, with only a small support circuit and a +5 volt power supply.

2.3.2. INTERFACE TO LOGGING DEVICE

The output voltage (Vo) and two reference (Vrl and Vrh) can be sent from the sensor circuit into a microcontroller ADC subsystem. The microcontroller can then be programmed to digitise and record temperature readings at regular intervals of time.

When the temperature monitoring system has been fabricated, then the two sensor subsystems can be fully evaluated and comparisons made with each other.

2.4 CHOOSING A SUITABLE MICROCONTROLLER CHIP

The following two sections of this chapter describe the differences between a single-chip microcomputer system and a single-chip microcontroller system. The descriptions should also show the suitability of a microcontroller for the two most important parts of the overall temperature monitoring system design: namely, the monitoring device and the data transfer

3.2.2. Single-chip

These are complete microcomputer systems on a single chip. They do not require any additional components other than a system clock signal to provide a single-chip solution to many of today's processing problems. Cooling (1991) states (page 20, `Software Design for Real-time Systems') that "Using a single-chip solution reduces the: package count, size and the cost of a system".

A microcomputer chip contains a CPU, memory (RAM and EPROM), timers, interrupt controllers, serial communication interface, parallel I/O ports and an external bus system. They are designed mainly as a processing device and can only handle serial, and parallel, I/O. Furthermore, they lack an analog subsystems; sample and hold and ADC,
They are not really designed, as data logging devices or control units, for real-time

3.2.3. Single-chip

Microcontrollers are derivatives of microcomputers; they are designed to provide all computing functions on a single chip. Cooling (1991) also states that "The interfacing hardware, internal register structure and the instruction set are optimised for real-time systems".

An initial task, at the time of commencing this Master's project, was a study of the most popular microcontrollers. Motorola, Intel and National Semiconductor were the names that were chosen as the manufacturers of the most popular microcontroller devices in the early 1990s. The data sheets and the manufacturer's support literature were studied in depth.

A comparative study of six series of microcontrollers was made, noting all their common features and their areas of specialisation. Two tables were made listing the most important features from the following six 8-bit microcontroller chips:

Motorola	M6801
	MC68HC11
Intel	8051
	8098
National Semiconductor	COP 800
	COP 888

(See tables 1.1 and 1.2)

This project revolves around the measurement and storage of temperature readings within the range of 35 to 40 degrees Celsius with a resolution of 0.1 degrees C. That means that there are 150 different possible values (note, 256 different values can be coded into 8-bits). Hence, an 8-bit analog to digital converter (ADC) can be used to input temp values and 8-bit memory locations can be used to store them. The total storage required for a complete 4 week test (28 days * 24 hours * 6 readings per hour) works out to be approximately 4Kbytes. Consequently, the author considered the available 8-bit microcontrollers; as it was believed that there was no need to look at the more powerful 16 and 32-bit devices.

A comparative analysis of the microcontrollers was made and a preliminary assessment of the requirements for a Temperature Monitoring Device (TMD) and a Data Transfer Device (DTD) were also made. This resulted in the following list of microcontroller requirements:

- A small portion of RAM (64 bytes) is required to hold the variables that are necessary for the control program.
- An 8K or 16K byte ROM, EPROM or EEPROM is required to permanently store the control program.
- A small portion of EEPROM is required to hold the program's parameters: namely, the device identification, a lookup table for the sensor readings, and the date and time of the start of the current test.
- A 4K or 8K byte RAM, or EEPROM, for the ongoing storage of data from the test.
- An 8-bit ADC to convert the analog data input from the sensor circuit into an 8-bit digital form
- A serial communication sub-system, ideally asynchronous, to upload the data to the data transfer device, or to the processing system (the PC system).
- An interface is required that can detect edges of pulses derived from the push buttons. These edges are necessary to activate the various functions of the data monitoring device.
- A timer sub-system is essential to enable precise increments in time to be measured. This allows data to be read at regular intervals of time, say every 10 minutes.
- A low power consumption is necessary for the data monitoring device. Ideally the microcontroller should also have pins available for parallel communication between the data transfer device and a hexadecimal keypad, and a liquid crystal display.
- 8-bit internal data paths and an 8-bit CPU are required to process, store and transfer the data.

It can be seen from the tables 1.1 and 1.2 that the Motorola microcontroller MC68HC11 is the most suitable chip, as it meets all the selection criteria for both the data monitoring device and the data transfer device

In the final analysis it could be said that the Motorola MC68HC11 microcontrollers appear to be the most versatile single-chip devices on the market to date. They have 4 main modes of operation and, in addition, they have low power dissipation modes of operation. Although other makes of microcontrollers could possibly have been used for this project the MC68HC11 has all interfaces required including an ADC sub-system. The onboard EEPROM and supporting onboard boot loader ROM programs enable these microcontrollers to have their EEPROMs to be programmed in situ.

Another reason for choosing the MC68HC11 microcontroller is that Motorola, in Perth, provide excellent hardware evaluation equipment and excellent software support for this series of products.

TABLE 1.1 MICROCONTROLLER COMPARISONS

Manufacturer Part Number	Motorola M6801	Motorola MC68HC11	National Semiconductor COP 800
Internal Memory	RAM 192-256 bytes ROM 2 - 4K bytes EPROM 2-4K bytes	RAM 192-768 bytes ROM 4K-24K bytes EEPROM 512-2K	RAM 64-128 bytes ROM 4K
Address Bus size	(16-bit) 64K bytes	(16-bit) 64K bytes	(15-bit) 32K bytes
Registers	6	7 + 64 I/O registers	6*8-bit + PC
CPU size	8-bit	8-bit	8-bit
Serial I/O	Asynchronous (FD)	Asynch (FD) Synch (high speed)	Synchronous
Parallel I/O	13 - 29 bits	A 4 = 0/P, 4 = I/P B 8-bit O/P C 8-bit I/O D 8-bit I/O E 8-bit I/P	2 I/O 8-bits 1 I/P 4-bits 1 O/P 4-bits
Timers	16-bit registers one I/P capture one O/p compare	16-bit registers 1 counter 3-4 I/P capture 3-5 O/P compare	16-bit registers mode 1 PWM mode 2 counter mode 3 timer
Analog Interface	none	8-bit ADC 8 channels (S&H)	8-bit 8 channels
Interrupts	IRQ and NMI	17 sources	3 sources
Watchdog	no	yes	yes
Operating Modes	1	4	3
Clock-rate	1 - 4 M Hz	DC to 8 M Hz	1 M Hz
Power	1 W	50 mW	1 0uW,10 mW,50 mW

TABLE 1.2 MICROCONTROLLER COMPARISONS

Manufacturer	Intel	Intel	National Semiconductors
Part Number	8051	8098	COP888
Internal Memory	RAM 128 bytes ROM 2 - 4K bytes EPROM 4K bytes	RAM 232 bytes (reg) ROM/EPROM 8K	RAM 128 bytes STACK (RAM) ROM 4K bytes
Address Bus size	(16-bit + 16-bit) 64K prog + 64K data	(16-bit) 64K bytes	(15-bit) 32K bytes
Registers	48 * 8-bit registers	232 registers	6 * 8-bit + 15-bit PC
CPU size	8-bit ALU	16-bit ALU	8-bit ALU
Serial I/O	Asynchronous (FD)	Asynchronous (FD) high speed sync	Synchronous (Microwire)
Parallel I/O	4 * 8-bit I/O ports	2 * 8-bit ports 2 * 4-bit ports	L 8-bit I/O G 5 I/O,1 I/P,3 O/P I 8-bit I/P (Hi-Z) D 8-bit O/P
Timers	2 * 16-bit registers 4 operating modes: I/P capture, event pulse width, mark space	16-bit registers 4 timers 2 counters PWM	3 * 16-bit timers 2 timers support: PWM, Event counter, I/P capture.
Analog Interface	none	10-bit resolution Sample and Hold 8 channel (MUX)	8-bit resolution 8 channels
Interrupts	6 sources, 5 vectors	21 sources, 9 vectors	10
Watchdog	no	yes	yes
Operating Modes	3	2	3
Clock-rate	1 - 10 M Hz	6 - 12 M Hz	DC to 20 M Hz
Power	1.0 W	1.0 W	50 mW

2.5. THE MICROCONTROLLER SYSTEM DESIGN ENVIRONMENT

2.5.1. INTRODUCTION

When designing data logging or control systems that make use of a microcontroller one needs to consider the type of platform necessary to develop the software and to test the interfaces to the user's hardware. The microcontroller system design environment can be considered to be in two parts, namely: the software design environment, and the hardware design environment.

2.5.2 THE SOFTWARE DESIGN ENVIRONMENT

The approach used for this project was to produce a software developement environment arround the suite of MC68HC11 support programs provided by Motorola. These programs are designed to run on an IBM PC or any compatable machine with an MS-DOS operating system. The Motorola suite of programs enables :

- assembly language source programs to be assembled,
- the assembler to create a program listing showing the source code, the equivalent macine code values and any syntax errors,
- the syntax free machine code to be linked to memory locations, and
- Motorola S-records to be created. Thus, enabling the machine code to be transferred, from an IBM PC to the memory of a MC68HC11 microcontroller, via the BUFFALO monitor program.

Programs are also required to edit the source program and for the serial communication between the PC and the Motorola evaluation board. The programs needed for the PC are:

- **Microsoft's full-screen editor,**
- **Motorola's portable asembler,**
- **Ubuilds program to create S-records and**
- **a serial communications program.**

In addition, a microcontroller debugger is required:

- **the Buffalo monitor program in the MC68HC11 to accept S-records and commands to debug a user program.**

The four PC programs used for software development were packaged into an efficient environment by calling them from within an MS-DOS batch file (written by the author). The batch file invokes two macinecode programs. One that clears the screen and selects forground and background colours for text. The other allows the user to select menu choices from within a batch file. A TYPE command inside the batch file creates a menu on the screen (see appendix D and the diagram below). The PC screen showed the following menu:

Type	**`1'**	**for**	**EDITING**
Type	**`2'**	**for**	**ASSEMBLING**
Type	**`3'**	**for a**	**LISTING**
Type	**`4'**	**for**	**S-RECORDS**
Type	**`5'**	**for**	**COMMUNICATIONS**
Type	**`6'**	**for**	**MS-DOS**

Programs are also required to edit the source program and for the serial communication between the PC and the Motorola evaluation board. The programs needed for the PC are:

- **Microsoft's full-screen editor,**
- **Motorola's portable asembler,**
- **Ubuilds program to create S-records and**
- **a serial communications program.**

In addition, a microcontroller debugger is required:

- **the Buffalo monitor program in the MC68HC11 to accept S-records and commands to debug a user program.**

The four PC programs used for software development were packaged into an efficient environment by calling them from within an MS-DOS batch file (written by the author). The batch file invokes two macinecode programs. One that clears the screen and selects forground and background colours for text. The other allows the user to select menu choices from within a batch file. A TYPE command inside the batch file creates a menu on the screen (see appendix D and the diagram below). The PC screen showed the following menu:

Type	**`1'**	**for**	**EDITING**
Type	**`2'**	**for**	**ASSEMBLING**
Type	**`3'**	**for a**	**LISTING**
Type	**`4'**	**for**	**S-RECORDS**
Type	**`5'**	**for**	**COMMUNICATIONS**
Type	**`6'**	**for**	**MS-DOS**

2.5.3 THE HARDWARE DESIGN ENVIRONMENT

The hardware support supplied by Motorola consists of three types of evaluation boards, namely: the EVBU, EVB and the EVM (see Figure 6). Each one of these evaluation boards contains a microcontroller system with an embedded monitor program. The monitor program allows communication, via one or two serial ports, with an IBM PC development system. The IBM PC is used to develop the software that is to be downloaded into the microcontroller system memory. Then the PC system is used to communicate with the evaluation board during the debugging stages of the software and hardware.

A detailed description of the uses for these Motorola evaluation boards is given in Appendix E.

Each evaluation board contains:

- a microcontroller chip,
- components and chips to support the microcontroller, and
- a monitor program to assist with the debugging procedure.

The EVB and the EVM evaluation boards also have external memory, and buffers to protect the ports of the MC68HC11 when interfacing to the user's target circuitry.

Note, an EVBU evaluation board was used to develop the TMD and to test the control program. Whereas, an EVU evaluation board was used to develop the DTD and to test its control program.

MICROCONTROLLER DESIGN METHODOLOGIES

Several library searches were made, at Edith Cowan University, on the following four databases:

1. INSPEC,
2. IEEE PUBLICATIONS,
3. SCIENCE and TECHNOLOGY CD-ROM NETWORK,
4. ENGINEERING and APPLIED SCIENCE.

1. INSPEC

A search for the number of articles, with abstracts containing the following terms were made:

Methodology	AND	Software	743
Methodology	AND	System Design	99
Methodology	AND	Computer Systems	29
Methodology	AND	Microcontrollers	7
Microcontroller(s)			519
Microcontrollers	AND	JSP	0

2. IEEE PUBLICATIONS

A search for the number of articles, with abstracts containing the following terms were made:

Methodology	AND	Software	908
Methodology	AND	System Design	172
Methodology	AND	Computer Systems	59
Methodology	AND	Microcontrollers	4
Microcontroller(s)			294
Microcontrollers	AND	JSP	0

3. SCIENCE and TECHNOLOGY CD-ROM NETWORK,

COMPUTER SELECT (77,816 Articles from Computer Periodicals)

A search for the number of articles, with abstracts containing the following terms were made:

Software	AND	Methodology	723
Microcontroller(s)			346
Microcontroller(s)	AND	Methodology	15
Microcontrollers	AND	JSP	0

DISSERTATION ABSTRACTS ONDISK (1988 - 1995)

A search for the number of abstracts containing the following terms were made:

Methodology	AND	Software	602
Methodology	AND	System Design	111
Methodology	AND	Computer Systems	54
Methodology	AND	Microcontrollers	4
Microcontroller(s)			49
Microcontrollers	AND	JSP	0

4. ENGINEERING and APPLIED SCIENCE

(Australian Engineering Database)

A search for the number of articles, with abstracts containing the following terms were made:

Methodology	AND	Software	23
Methodology	AND	System Design	73
Methodology	AND	Computer Systems	30
Methodology	AND	Microcontrollers	0
Microcontroller(s)			5
Microcontrollers	AND	JSP	0

The conclusions drawn from all the aforementioned results were as follows:

- There was a great interest, during 1988 to 1995, in Software Design Methodologies.
- There was less, and still is less, of an interest in describing design methodologies in computer system design articles.
- Hardly any articles, that described microcontroller applications, gave information on their design methodologies (7 out of 519, 4 out of 294 and 15 out of 346). Note, 4 out of 49 abstracts from dissertations mentioned their design methodologies.
- Finally, and most importantly to this thesis, no articles or dissertations describe how a JSP methodology could successfully be used to help design a microcontroller-based system.

Note, that the few microcontroller design methodologies that were discussed fell into two categories:

methods describing the internal design of custom-made microcontrollers were given or

the methods of designing systems using standard microcontroller chips and the associated software were detailed.

CHAPTER THREE

3 HARDWARE DESIGN

3.0. OVERVIEW OF THE TEMPERATURE MONITORING SYSTEM

The temperature monitoring system (TMS) comprises of seven main parts:

- a person under test,
- a temperature sensor,
- a Temperature Monitoring Device (TMD),
- a Data Transfer Device (DTD),
- a Data Processing System (DPS),
- an output device and
- a researcher. (see figures 7 and 11)

A brief description of each part of the system is given below and a more comprehensive description of the whole system is given in chapter 4.

- The person under test provides a source of temperature between the range of 35 through to 43 degrees Celsius.
- The temperature sensor is expected to monitor the person's temperature with a resolution (and relative accuracy) of 0.1 degrees Celsius.
- The temperature monitoring device is a miniature data logging system that: receives input from the sensor, is capable of storing four thousand temperature readings and is able to download the data at the end of a test, to a DTD or DPS.

- The main functions of the data transfer device are: to keep the correct time and date, to be able to download time date and a start signal, to the TMD, in order to begin a test session. Then later, upload the information from a test in order to pass it on to the DPS. Note, this small device enables the remote use of the TMD; away from an office environment.

- The data processing system is used by the researcher to: upload information from a test, check the information on a VDU screen and store data from a test on a file. Then later, the researcher can process the data in order to produce graphical information.

- The output device enables hard copies of results to be made.

- The researcher is responsible for organising tests and any programs that process the results.

See figure 11 for a graphical representation of the data flow within the system.

The following four major sections of this chapter describe the steps taken during the hardware design of the temperature sensor, the TMD, the DTD and the DPS.

3.1.1. THE LM 35 TEMPERATURE SENSOR

IC Temperature Sensor

The IC temperature sensors are based on the diode voltage being temperature dependant. One version (the AD590) has a current output proportional to absolute temperature. The sensor is insensitive to the voltage applied across it and can be used even with long lead wires. Another version (the LM 35) has an output voltage proportional to temperature.

LM 35 IC Sensor

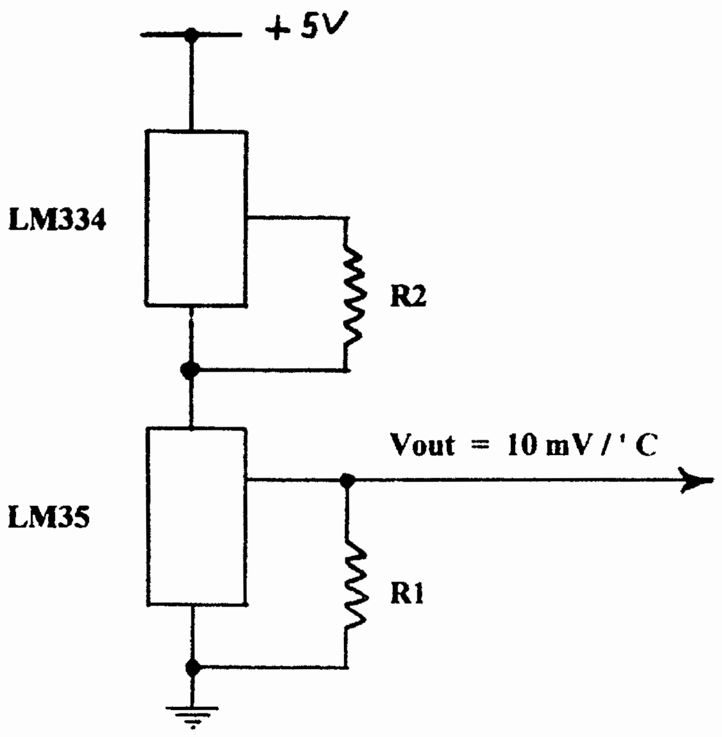
The LM 35 series of integrated circuits are precision temperature sensors, whose output voltage is linearly proportional to Celsius temperature. The user is not required to subtract a large constant voltage from its output in order to obtain a convenient degree Fahrenheit scaling. [RS Components data sheet 8307]

The LM 35's low output impedance, linear output and precise inherent calibration make interfacing to readout or control circuitry especially easy. The IC draws only 70 uA from its supply, it has very low self heating, less than 0 . 2' C in still air. No trimming is required to gain an accuracy of + or - 0 .1`C at room temperature. The LM 35Z chip is rated to operate over a -40 to +110 `C temperature range.

Features

• wide temperature range	-40' C to +110' C (CZ version)
• accurate	0.25' C at room temperature
• linear output	+/- 0 . 1' C typical
• low self heating	0.08' C typical
• output impedance	0.1 ohm at 1 mA
• output voltage	10mA per degree Celsius
• supply voltage	+35 to -1.0 volts

3.1.2. THE LM 35 / LM 334 TEMPERATURE SENSOR CIRCUIT



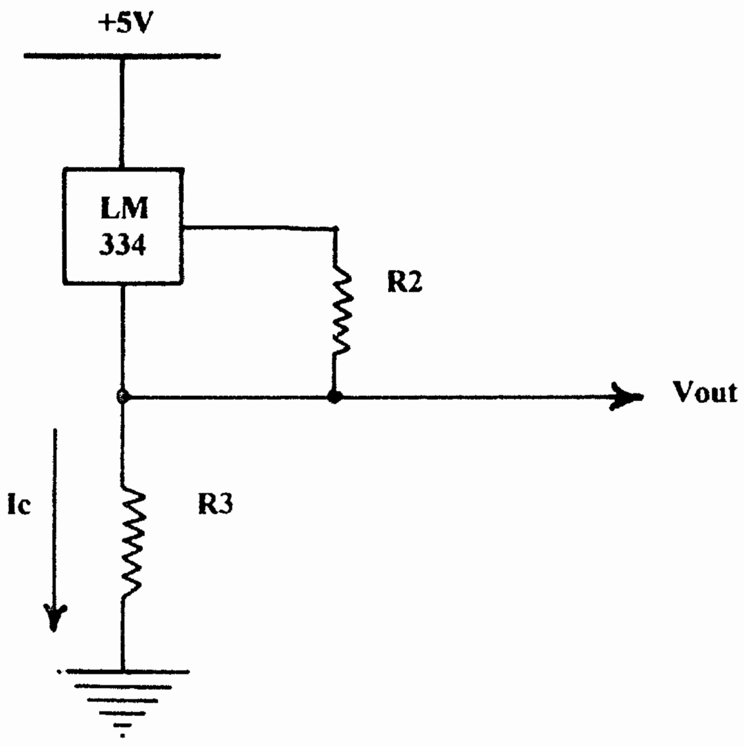
Selecting the value for resistor R1

A test of the circuit was performed in order to select the value of resistor R1.
R1 determines the nominal output voltage at room temperature

R1 (ohms)	Vout (volts)
56	2.20
68	2.22
180	2.32
270	2.40
330	2.45
560	4.25
680	4.27

Selecting the value for resistor R2

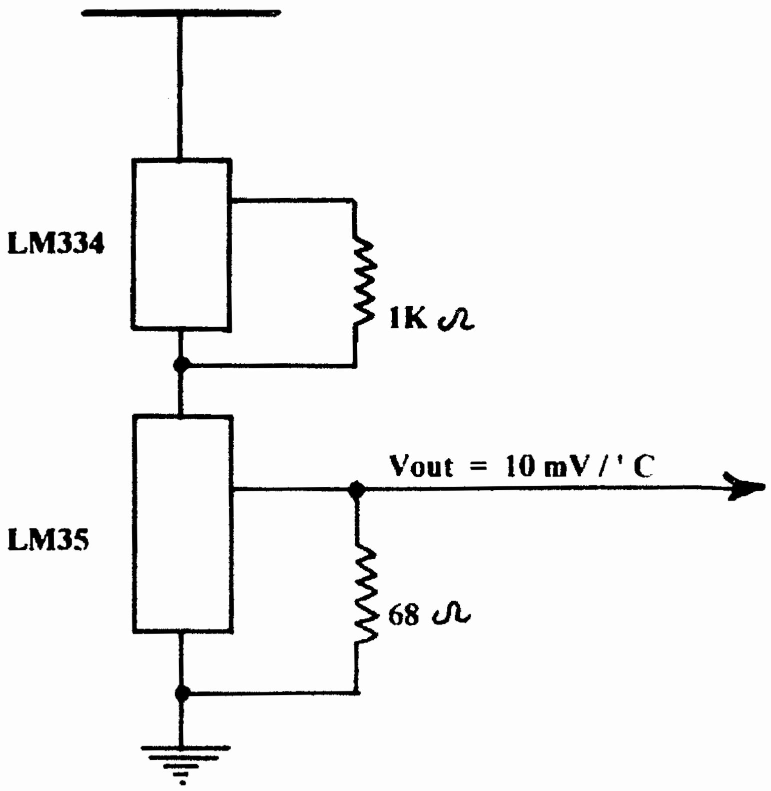
The LM 334 is a constant current source IC. The magnetude of the constant current is determined by the size of the resistance of the external resistor R2. The resistance for R3 was chosen to be 47K ohms as this approximately represents the resistance of the temperature sensor LM35. The test circuit shown below was constructed in order to select a suitable value for the constant current.



Results

R2	R3	Vout	Ic (Vout / R3)
39K	47K	0.080V	0.001uA
22K	"	0.150	0.003
10K	"	0.318	0.007
5.5K	"	0.587	0.013
2.2K	"	1.540	0.046
1K	"	3.41	0.072
.56 K	"	4.26	0.090
33 K	"	4.29	0.091

3.1.3. THE LM 35 / LM 334 SENSOR CIRCUIT RESULTS



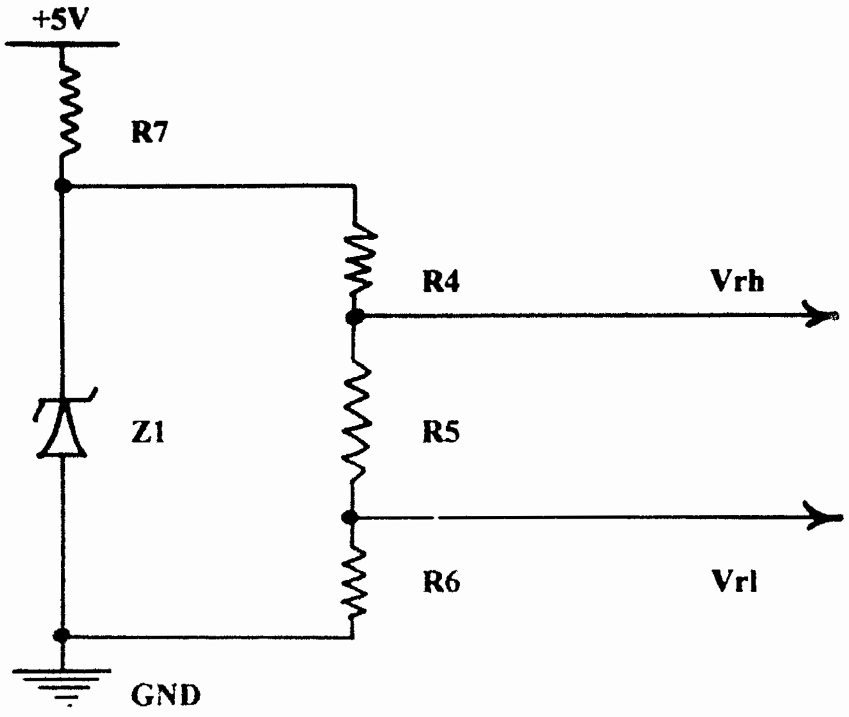
Results Output voltage versus Temperature

In the test circuit shown above, the 1K ohm resistor was chosen so that only a small constant current (nominally 70uA) flows through the temperature sensor circuit. This reduces the drain on the supply which is an important criterion for this project. The 68 ohm resistor was chosen so that the nominal output voltage at room temperature was around 0.2 volts .

Temp (' C)	Output (mV)	Temp (' C)	Output (mV)
4	2	33.4	288
12	82	40.0	355
14.6	109	42.5	376
15.8	112	46.0	408
16.0	122	48.0	452
18.0	145	51.5	464
21.0	180	53.0	480
23.0	186	56.0	511
27.3	212	57.0	520
30.0	254	70.0	660
32.0	280		

3.1.4. THE VOLTAGE REFERENCE CICUIT CALCULATIONS

It was decided that the voltage referece circuit shown below was to be used to provide the voltages Vrh and Vrl for the microcontroller ADC subsystem.



To Calculate the Resistor Values for the Voltage Reference Circuit (R6, R5 & R4)

given

Vref	=	4.00 volts	
Vrh	=	0.46volts	(nominal output for 50' C)
Vrl	=	0.25 volts	(nominal output for 30' C)

calculate

Voltage across R6	=	Vrl	=	0.25 volts	
Voltage across R5	=	Vrh - Vrl	=	0.46 - 0.25	= 0.21 volts
Voltage across R4	=	4 - Vrh	=	4.00 - 0.46	= 3.54 volts

Vz = 4 = 0.25 + 0.21 + 3.54

Since the same amount of current flows through R6, R5 and R4 then the resistor values will be in the same ratio as the voltages: 0.25 : 0.21 : 3.54 Choosing resistor values with a 1% resistor tolerance and a temperature coefficient of 100ppm in the ratio (2500 : 2200 : 3600) will output the following reference values:

Vrh = (R6 + R5) * 4 / R6 + R5 + R4 = 4700 * 4 / 40700 = 0.462 volts

Vrl = (R6 * 4) / R6 + R5 + R4 =2500 * 4 / 40700 = 0.246 volts

Hence:

R1	=	68	ohms	
R2	=	22K	ohms	
R3	=	2.7K	ohms	
R4	=	36K	ohms	
R5	=	2.2K	ohms	
R6	=	2.5K	ohms	
R7	=	560	ohms	(to provide a 1.7 mA current)
Z1	=	ZN REF 040		

3.1.5. THE YSI PRECISION THERMISTOR

Thermistors

Darold Wobschall (1987) states that basically a thermistor is a resistor with a high temperature coefficient. It is a semiconductor that is found in various geometrical configurations to which leads are attached. In fact they are found in a wide variety of shapes and sizes (down to microscopic sizes) are possible.
They have a negative temperature coefficient; their resistance decreases with increasing temperature. The resistance-temperature variation is non-linear; over most of its range the resistance decreases exponentially with temperature.

Tompkins and Webster (1988) state that although thermistors are not highly precision sensors, they are used because they have a low cost, high sensitivity, ease of readout and small size. A convenient configuration can also be found to suit a particular application. For example, bead, disc, screw-in, diode and thin film versions can be bought.

Thermistors are affected by internal heating (power dissipation) caused by the voltage applied to the thermistor from the readout or converter unit.

The YSI 44002 Precision Thermistor

The YSI 44002 Precision Thermistor has the following specifications:

• resistance	300 ohms at 25 degrees Celsius.
• time constant	10 seconds in air, 1 second in stirred oil. (time to reach 63% of a new reading)
• power dissipation constant	1 mW in air, 8 mW in oil (power required to raise the temperature 1 degree above the ambient temperature)
• Stability	The manufacturers state that these devices have a proven long term stability if operated around 25' Celsius. For long term use at higher temperatures the manufacturers recommend a 3000 ohm version of this device.

3.1.6. THE YSI 44002 CALIBRATION TABLE

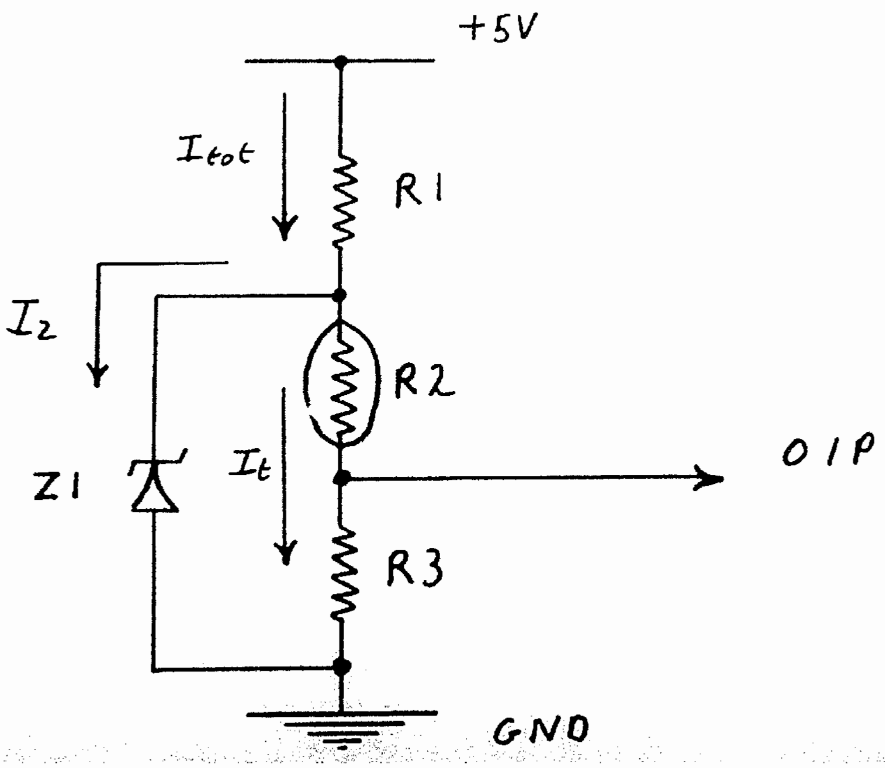
TABLE 2 THERMISTOR CALIBRATION TABLE

TEMPERATURE	RESISTANCE	TEMPERATURE	RESISTANCE
30	252.4	40	181.2
31	244.0	41	175.5
32	235.9	42	170.0
33	228.1	43	164.7
34	220.6	44	159.6
35	213.4	45	154.6
36	206.5	46	149.9
37	199.8	47	145.3
38	193.4	48	140.9
39	187.2	49	136.6

The calibration table values that are shown above came with the YSI 44002 thermistor.

These calibration values were used to calculate the expected output voltage range when the YSI 44002 thermistor is subjected to a temperature change from 30 to 50 degrees C.

3.1.7. THE YSI 44002 SENSOR CIRCUIT



3.1.8. THE YSI 44002 CIRCUIT DIAGRAM CALCULATIONS

To Calculate the Change in Resistance of the Thermistor

The temperature sensor was designed to measure temperatures within the approximate range of 20' C to 50' C. The change in resistance of the YSI 44002 thermistor was calculated by inserting the resistance values for 20' C and 49' C into the following formula:

Rmax (20' C) - Rmin (49' C) = 252.4 - 136.6 = 117.8 ohms

Therefore **R2max = 252.4 ohms,** **R2min = 136.6 ohms**

To Calculate the Value for Resistor R1

Given

Supply voltage		=	5 volts
Reference voltage for	Z1	=	4 volts
Current required for	Z1	=	0.3 mA
Thermistor resistance	R2min	=	136.6 ohms
Consider resistor	R3	=	2.7 K ohms

Calculate

It = V/R =4 / (R2 + R3) = 4 / (136.6 + 2700)

= 4 / 2836.6 = 1.4 mA

Current through R3 Itot = Iz + It

Itot = 0.3 mA + 1.4 mA = 1.7 mA

Voltage across R1 = 5 - 4 = 1 volt

Value for R1 = (R=V / I)= 1 / 1.7 = 560 ohms

Therefore = **R1 = 560 ohms**

High Output Voltage Calculation

Consider the case (YSI Sensor Circuit) where R3 = 2.7 K ohms

Vout = R3 * Vz / R2 + R3

Vout = (R3 * 4)/ (R2min + R3) =(2700 * 4) / (136.6 + 2700)

Vout = 3.81 volts

Low Output Voltage Calculation

Consider the case (YSI Sensor Circuit) where R3 = 2.7 K ohms

Vout = R3 * Vz / R2 + R3

Vout = (R3 * 4)/ (R2max + R3) =(2700 * 4) / (252.4 + 2700)

Vout = 3.65 volts

To Calculate the Resistor Values for the Voltage Reference Circuit (R6, R5 & R4)

given

Vref = 4.00 volts

Vrh = 3.81 volts

Vrl = 3.65 volts

calculate

Voltage across R6 = Vrl = 3.65 volts

Voltage across R5 = Vrh - Vrl = 3.81 - 3.65 = 0.16 volts

Voltage across R4 = 4 - Vrh = 4.00 - 3.81 = 0.19 volts

Vz = 4 = 3.65 + 0.16 + 0.19

Since the same amount of current flows through R6, R5 and R4 then the resistor values will be in the same ratio as the voltages: 3.65 : 0.16 : 0.19 Choosing resistor values with a 1% resistor tolerance and a temperature coefficient of 100ppm in the ratio (36000 : 1600 : 1800) will output the following reference values:

Vrh = (R6 + R5) * 4 / R6 + R5 + R4 = 37600 * 4 / 39400 = 3.82 volts

Vrl = (R6 * 4) / R6 + R5 + R4=36000 * 4 / 39400 = 3.65 volts

Hence:

R1	=	560	ohms
R2	=	YSI 44002	
R3	=	2.7K	ohms
R4	=	1.8K	ohms
R5	=	1.6K	ohms
R6	=	36K	ohms
Z1	=	ZN REF 040	

Conclusions

Prototypes of both circuits were constructed and tested in laboratoty conditions. The results indicated that either circuit could be suitable for monitoring human body temperature.

3.2. DESIGNING THE TEMPERATURE MONITORING DEVICE

3.2.1. THE FUNCTIONS REQUIRED FROM THE TMD

The first stage of the design involved the use of a top-down design approach to the whole system. The system was subdivided into five main parts, as shown below:

(i) the temperature sensor,

(ii) the temperature monitoring device (TMD),

(iii) the data transfer device (DTD),

(iv) the data processing system (DPS), and

(v) the output device (printer/plotter). (see Figure 7)

The second stage of the design was to analyse each part of the system , in turn.

The Temperature Sensor

The temperature sensor must be capable of measuring temperature from a suitable body site. The temperature must be used to indicate the core temperature of the body in question. The temperature sensor must be able to respond to changes in temperature within the range: 35.0 degrees Celsius to 43.0 degrees Celsius, with steps of 0.1 degrees.

The Temperature Monitoring Device

The functions of the temperature monitoring device were listed as follows:

- to receive the signals from the sensors.

- to condition the signals from the sensors so that they are in a form suitable for an ADC. The signals have to be within an expected range of values, with reference to two fixed voltages (voltage reference high (Vrh) and voltage reference low (Vrl)).

- to save the temperature measurements in a digitised form at regular intervals in time (say, every ten minutes). A semiconductor read/write memory capable of storing 4000 readings would be required to store a months supply of data.

- to respond to push-button commands that call up service routines.

- to upload the start of test time and date from a DTD and hence, start logging data..

- to download an identification label and time related data to a `transfer device' or to a personal computer system:

A top-down approach to the design of the temperature monitoring device revealed that the following parts were necessary.

(i) a signal conditioning unit for the sensor readings,

(ii) an analog to digital converter (ADC) to digitise the sensor readings,

(iii) a storage device for an ident label, time and temperature readings.

(iv) a real-time clock system, or a means of entering the start of test time.

(v) a serial communications unit,

(vi) a means of setting the time of day in the real-time clock system,

(vii) a means of knowing that the monitoring device is functioning correctly,

(viii) a unit to control the seven units mentioned above,

(ix) a power supply unit.

The first five parts of the temperature monitoring system and the eighth part are complete sub-systems. Integrated circuits are available for each of these separate subsytems. For example:

- An operational amplifier for the signal conditioning unit.
- An 8-bit ADC chip to digitise the sensor signals.
- A real-time clock chip so that the temperature readings could be aligned to the time of day.
- A random access memory to store an identification label, time and temperature data.
- A serial communications chip, possibly a Universal Asynchronous Receive/Transmit (UART) chip.
- The sub-system that controls the other sub systems could also be a special purpose chip.

Whilst considering the features of the proposed design, the overriding characteristic of the system had to be remembered. The temperature monitoring device has to be as small as possible. So a multi-chip design, although feasible, would be rather large, and not meet the small device criteria and would also make the power supply module too large.

However, further reduction in size, power and weight are possible by using programmable multi-function devices on the market that are designed for real-time applications. A single chip microcomputer system: and, in particular, a microcontroller chip would fit our application's functional needs. (see Figures 4, 5 and 8)

From the hardware point of view, a microcontroller's internal computing equipment includes various input/output interfaces. Microcontrollers may have:

- 8-channel analog multiplexing,
- a sample and hold module for analog signals,
- an analog to digital converter sub-system,
- a pulse width modulation unit
- an independent timer sub-system,
- a fast synchronous serial sub-system, and
- an output compare sub-system.

Microcontrollers also differ from microcomputers in other ways: They may have

- four modes of operation:
- a special test mode,
- a special bootstrap mode,
- a single chip mode, and
- an expanded multiplexed mode of operation.

Taking into account the features mentioned above, at the time of designing the TMS, the MC68HC11 microcontroller was considered to be the most suitable device to use as the control centre of the TMD. In particular, the MC68HC11 was the only microcontroller (known to the author) that had an internal ADC subsystem. (see section 2.4. and tables 1.1. and 1.2.)

3.2.4. THE MOTOROLA MC68HC11 SERIES OF MICROCONTROLLERS

The MC68HC11 series of single chip microcontrollers are available in either a 52 pin plastic leaded chip carrier (PLCC) package or a 48 pin dual-in-line package (DIP).

The MC68HC11 series contains the sub systems that are essential to the design of the temperature monitoring device (see Figure 4 and 5). They contain:

- memory (RAM, ROM and EEPROM),
- an 8-bit ADC sub-system,
- a serial communication interface,
- a serial peripheral interface,
- a timer/counter subsystem
- hardware interrupt logic, and
- the MC68HC11 CPU.

The only function not supplied is the sensor's signal conditioning and a real-time clock.

The real-time clock chip can be maintained by the data transfer device. The data transfer device could download the correct date and time to the monitoring device just prior to a data logging session. Hence, the temperature monitoring device need only read data at precise fixed time intervals (say, every 10 minutes), prompted by software monitoring the timer sub-system counters.

Another feature to note is that the MC68HC11 family of 8-bit microcontrollers have an address space of 64K bytes, which is large enough to store control programs and all the required data from a monitoring session (see Figure 5).

Although the data path of the microcontroller is 8-bits, but the chip is capable of 16-bit arithmetic. Hence, it was decided that the MC68HC11 microcontroller was an ideal device to use as the central component of out TMD design. (see Figure 12). Push buttons, resistors, LEDs and a +5 volt power supply were included into the TMD system to complete the design.

The push buttons enable the user control the data logging system. The device can be: started, stopped and show it's status.

3.3 DESIGNING THE DATA TRANSFER DEVICE (DTD)

The reason for designing a small, portable, data transfer device (DTD) are as follows:

- A device is required to download the date and time and a signal to a TMD in order to start a test.
- To enable the remote use of a TMD away from an office environment.
- To house a 4K byte EEPROM chip in a low-insertion socket. The DTD can then be used to upload information from a TMD, at the end of a test (into its EEPROM). The EEPROM can then be taken out from the socket and posted from a remote site. Later, the EEPROM can be inserted into another DTD (by a researcher) in order to download the data into a DPS.
- Note, The DTD could be produced at a much lower cost than a laptop computer system. It would be much smaller in size and therefore be more portable.

The data transfer device has to perform the following functions:

- to be able to store at least 4K bytes of data.
- to upload the data from the temperature monitoring device.
- to download the date and time to the temperature monitoring device.
- to download the data to the data processing system.
- to maintain the correct date and time.
- to be able to change the date and time.
- to be able to respond to key-strokes from a hexadecimal keypad.
- to be able to display the data, date, time and a command menu.

Whilst considering the design of the data transfer device, an investigation took place in order to gain knowledge into interfacing to a: liquid crystal display (LCD) module, hexadecimal keypad and serial communication channel. It was found that the MC68HC11 could interface to a LCD module and a hexadecimal keypad quite easily. Hence, the MC68HC11 could also be used for the control centre of the DTD. Although other microcontrollers could have been used, the MC68HC11 was chosen in order to be able to use the same hardware development environment and be able to use the same software development tools. So, with the MC68HC11 microcontroller in mind, and using a top-down design approach, the data transfer device was seen to have consisted of the following modules:

- A hexadecimal keypad to input changes to the date and time, and to input commands to drive the device.
- A liquid crystal display (LCD) module with a a 4-line 20-character screen.
- A real-time clock chip or a means of entering and viewing the date and time.
- A read/write memory chip consisting of at least 4K bytes of storage.
- A serial communications port for uploading and downloading information.

- A MC68HC11 microcontroller that contains a serial communications interface, parallel I/O ports and a means of controlling the system components.
- A power supply unit (batteries).
  (see Figures 9 and 13)

3.4. DESIGNING THE DATA PROCESSING SYSTEM

3.4.1. The Functional Requirements

The data processing system has to perform the following functions:

- to upload the identification label, the date and time, and the data, from either the data transfer device or the temperature monitoring device itself.
- to store the data on a secondary storage file.
- to process the data and produce a time-temperature graph.
- to be able to display the results of the data processing on a visual display unit (VDU) and also on a printer capable of plotting graphics. The output must be in a text and a graphical form.

The functional descriptions mentioned above could all be performed on almost any personal computer system that has a serial communications port and a graphics adaptor card for the VDU.

So the hardware requirements for the data processing system are as follows:

A personal computer containing the following parts:

- a graphics adaptor,
- a serial port,
- a printer port, and
- a secondary storage device

The software required to perform the tasks mentioned previously are as follows:

- A communications package that enables uploading to a secondary storage file

- Any commercial spreadsheet, or a user designed applications package written in a high-level language.

The Output Device

Any make of printer that is capable of producing graphics with a resolution of 180 dots-per-inch, or greater, and is able to interface with the data processing system mentioned above.

(see Figure 10)

CHAPTER FOUR

4. SOFTWARE DESIGN

(The Software Requirements Document)

4.1. INTRODUCTION

This chapter describes the temperature monitoring system (TMS) from a Software Engineer's point of view. The information in this section describes the functions of the system, and how a programmer may write, debug and test each module of the programs. Sections 4.2. and 4.3. describe the functions of the hardware and the flow of data and commands between the three main parts of the TMS, namely:

The Temperature Monitoring Device	(TMD)
The Data Transfer Device	(DTD)
The Data Processing System	(DPS)

Sections 4.4. to 4.10. describe the three main parts of the system in the following ways:

- their functional requirements,
- the names of the data types to be used,
- the non - functional requirements,
- how structured analysis is used to create data flow diagrams (DFD),
- how transform analysis is used to transform DFDs into Jackson Structured Diagrams (JSD),
- how JSDs are used to develop programs and
- information that will enable maintenance and testing of the software.

Software Engineering practices were employed during the design of this project. A variety of methods of program design were considered, which lead to a Jackson Structured Programming (JSP) methodology being investigated and used as it appeared, to the author, to be the most appropriate method for real-time applications. A JSP methodology is an approved approach to many real-time applications as it ensures that the final product is a well engineered solution that meets the principles of software engineering.

Products designed using sound software engineering principles have:

- maintainability (perfective, adaptive and corrective),
- language independent design,
- modu!arised testing (black-box, white-box) and
- verification and validation.

STRUCTURED PROGRAMMING METHODOLOGIES

The Need For Analysis Techniques

There are several analysis techniques used throughout the world today. They support:

- a hierarchical representation of a system,
- each carefully considers external and internal interfaces and
- each provides a foundation for design, implementation and testing steps.

The need for a systemised method for developing software is best described by Alvey (1986). He found that whilst programming standards are considered to be a good thing, in general they are ignored and that one way to prevent this would be to enforce them automatically by means of a software tool.

Pressman (1992, p. 267) states that any requirements analysis method combines a set of distinct heuristics and a unique notation to analyse information, functions and behaviour of a computer-based system. Through the application of the fundamental analysis principles each method creates a model of the problem and a required solution.

Analysis Techniques

A range of analysis techniques exist. This section describes the following three main areas of design:

Structured analysis is a model building activity which illustrates, the flow of data and control. It depicts the essence of what must be built. DeMarco (1979, p. 15) establishes the primary goals of an analysis method as:

- **the product of an analysis process must be maintainable,**
- **graphics must be used** wherever possible,
- there is a need to differentiate between logical and physical considerations,
- and there is a need to keep track of and evaluate interfaces.

Structured analysis is an information and content modelling technique where circles, squares, arrows and sets of parallel lines represent: transforms, external entities, inputs and outputs, and storage components of a system.

The advantages of using structured analysis design are as follows: It is a systematic method, it is very graphical, it is easy to follow information transforms throughout the various stages of the design process and in this research work, it is ideal for real-time system design.

The main disadvantages of structured analysis design could be that: it is an iterative process, and that structure clashes cause problems when converting data structures into a program structure.

The main contributors to the development of structured analysis techniques are: Jackson, Hoare, Orr, Warnier and Yourdon.

Object-oriented analysis (OOA) is making slow but steady progress as a requirements analysis method in its own right and as a complement to other analysis methods. Pressman (1992, p. 239).

Object-oriented analysis is based upon objects and attributes and classes and members, rather than data flow and structured analysis. Object-oriented techniques allow designers, programmers and users to view concepts as a variety of units or objects that fit into a hierarchy of different components or structure. By using object-oriented techniques designers can represent neatly the relationship between: components, objects, tasks to be performed and conditions to be met.
The code can be reused and easily changed by subsequent designers. The three main elements of object-oriented techniques are:

- data encapsulation,
- inheritance and attributes and
- polymorphism (overloading of operator names).

The main advantage of object-oriented design is that it enables designers to build a system based upon: abstraction, information hiding and modularity; without complexity or compromises.

The main disadvantages of object-oriented design are: that it is not really language independent and that object-oriented compilors (such as Ada or C++) are not always available for microcontroller development systems.

The main contributors to the development of object-oriented design processes are: Booch, Coad et al, Meyer, Shlaer et al and Wiener et al.

Formal specification techniques are also being examined today. Formal methods enable a software engineer to specify, develop and verify a computer-based system by applying a vigorous mathematical notation. Pressman (1992, p. 288)

Formal specification languages employ three primary components:

- syntax,
- semantics and
- a set of relations.

The syntax includes variables such as x, y and z and logic symbols such as which represent: all, there exists, not, and, and or.
The semantics indicates how the language represents system requirements.
The relations define rules that indicate which objects properly satisfy the specification.

Pressman (1992, p. 287) informs us that the use of a formal specification language provides a means of specifying a system so that consistency, completeness and correctness can be assessed in a systematic fashion.

The advantages of a formal approach to system design are: that it is easy to create design tools and tools for testing a design. Hence, the consistency, completeness and correctness of a system can easily be assessed in a systematic fashion.
The disadvantages could be that it is not easy to get people interested in formal approaches to design because: it is difficult to learn/teach, it appears to be complex, it is not very visual and it uses unfamiliar notation.

The main contributors to the development of formal specification languages are: Hall, Wing and Woodcock.

THE JSP METHOD OF DEVELOPING SOFTWARE

Why Choose JSP

The programming methodology adopted in this dissertation is Jackson Structured Programming (JSP). Reasons for adopting a JSP methodology include:

JSP is not a programming language; it is a method for developing programs. In fact it is language independent Cameron (1989, p. 15).

Cameron (1989, p. 19) states that "commercial programmers are often surprised, when they first translate a Jackson Structure Diagram (with functions allocated) into actual programming code, how close to the finished program they were".

Cameron (1989, p. 5) informs us that JSP in particular is very good , as it allows the same notation and techniques to be used at different times throughout a design procedure. For example, structure diagrams using the same notation can be used to describe:

- the ordering of events,
- the ordering of data components and
- the program itself.

This makes life easier during testing and maintenance of programs.

Formulating a JSP Methodology

Before deciding a JSP methodology for this Masters Thesis several JSP methodologies were examined. For example:

> Cameron (1989, p. 11) describes a JSP process as having four major steps:
>
> 1. Draw structure diagrams to describe each of the data streams: input to or output from a program.
> 2. Merge these data structure diagrams into a single structure diagram, a program structure diagram.
> 3. Make a suitable list of executable operations from the programming language to be used. Allocate the operations, one by one, into the program structure diagram.
> 4. Convert the program from the diagrammatic representation into a textural form and add conditions to iteration and selection components.

Bell et al (1987) states that "Jackson's data structured design method is dramatically different from other approaches to programming design. It is the most systematic method in existence. The basic idea behind JSP is that the structure of a program should match the structure of the data types it is going to act upon and the I/O mechanism used. The methodology by Bell, Morley and Pugh (1987) [page 52, is summarised in a similar way to Cameron's.

Mohri and Kikuno (1991) formulated thirteen steps (S1 through to S13) which they adopted as a JSP development process. The details are specified as follows:

step S1	(Understanding program specifications)
step S2	(Formulating diagrams for input and output data structures)
step S3	(Formulating program structure diagram)
step S4	(Enumerating variables)
step S5	(Enumerating operations)
step S6	(Allocating operations to program structure)
step S7	(Optimising the program structure)
step S8	(Design review)
step S9	(Coding)
step S10	(Code review)
step S11	(Preparing test data)
step S12	(Unit test)
step S13	(Integration test)

The following JSP development process was adopted as it appeared to the author to be the most suitable method of designing small real-time microcontroller systems.

1. The functional requirements, of each part of the system, were listed.

2. The names of the data types to be used were listed, for each part of the system.

3. The non functional requirements, of each part of the system, were described fully.

4. A structure analysis technique was used to produce data flow diagrams DFDs of various levels of the system.

5. A transform analysis technique was used to convert the DFDs into program structure diagrams.

6. A list of elementary functions and subroutine calls was made. Each function of subroutine call was given a unique number that is associated to the current program structure diagram.

7. The numbers representing the elementary functions or subroutine calls were inserted into the applicable program structure.

8. A list of conditions relating to iterative processes or selections were made, for each part of the program. Each condition was given a label (a unique number preceded by the letter `C'.

9. The labels representing conditions were inserted into the appropriate place in the program structure diagram.

10. The program, for each part of the system, was converted from the diagrammatic representation into a textural form. The program code included: elementary functions, subroutine calls and conditions to iterative and selection components.

11. Finally, for each part of the system, maintenance and testing information were described.

Structure Clashes

Hashimoto and Okamoto (1990) described a structure clash as: one of the main concerns in JSP. The clashes occur between the processing of input data and the processing of output data. This happens when the two or more data structures involved in a problem cannot be mapped onto a single program structure. This is due to a fundamental incompatibility between input data structure and output data structure.

The solution to structure clashes is quite simple; two programs need to be designed instead of one. The first program organises the input data into a form used by the data being output by the second program.

Testability And Maintinability Of JSP

Cameron (1991, p. 27) stated the following :

The structure of a program should be based on the structure of the underlying problem. Hence, the component of a problem should recognisably map directly onto the components of a program. Any important object of a problem must have a corresponding program component. Therefore, naming the intermediate components of a data structure becomes important. Correct program structure is essential in order to make the subsequent testing and maintenance easier. The JSP idea of correctness is achieved by having correct procedures and also a structure to match the problem.

Roper and Smith (1988) state that: "the problem with testing programs is that it often involves more work than designing and writing the program in the first place. Whereas, there is an inherent testability with a JSP design process".

COMPARISONS WITH OTHER METHODS OF DESIGN

Cameron (1991, p. 7) informs us that it would be natural to expect a concise, coherent comparison of JSP with other software development methods and to be able to describe those unique features worthy of attention. Unfortunately, the field of method comparison is somewhat problematic. There have been a number of valiant attempts, but none seem entirely successful.

For example, Rozman (1989) set up an experiment using a small number of post graduate electrical engineers. Two methodologies were compared: System Analysis - System Design and Jackson System Development. Their conclusions stressed that it was not the intention of their research work to compare the two methodologies and suggest a valid choice. However, their study did highlight the receptability of a methodology by electrical engineers who had previously poor knowledge of software engineering practices. They also emphasised that testing of other profiles of specialists may lead to absolutely different conclusions.

OTHER WORKS ON THE JSP METHODOLOGY

Roper (1988) identified the need for a more formal approach to software testing, and produced a methodology for testing programs constructed using Jackson structured programming techniques. Algebrate expressions were generated, from information contained in a JSP structure chart, and used in a novel structured testing method.

Roper and Smith (1988) stated that "they have developed a novel testing methodology which exploits the inherent testability of the JSP design process. which also integrates fully with that process".

Thompson (1990) research has produced a tool which can be used to check source code, which has been implemented from JSP designs. John Barrie Thompson concluded, that a Quality Assurance Tool must be a worthwhile exercise as this does ensure that implementation standards cannot be ignored.

Edwards (1990) describes how her research focused upon the development and assessment of a systematic and formalised interface between Structured Systems Analysis And Design (SSADM) and JSP. Her methods encompass the entire software life cycle.

Davies (1987) informs us how a series of computer assisted tools including a program structure generator were integrated to form a computer aided program design system at UMIST. A method was developed by which two Jackson data structures may be merged to produce a Jackson program structure. C. G. Davies research also included an investigation into program maintenance with respect to the development of process structure and established rules to govern possible designs.

CONCLUSIONS

A JSP methodology was chosen for this thesis because it provided a highly systematic approach to software design. It is a methodology that is loosely defined, which enables it to have extensions that make it useable in a variety of situations. For example, JSP can be used for designing:

- data processing systems,
- large real-time systems and
- in this case, small microcontroller systems.

The initial stages of design are language independent, it is only when you come to the stage for allocating elementary functions that you may need to consider the language that will be used to implement the design.

A JSP methodology creates a very graphical hierarchical solution to a problem. The control and decision making structures are created at the top of the tree structure; the functions that input information, process information and output information are seen at the lower extremities of the tree structure.

JSP provides a high degree of modularity into a solution of a problem. A design consists of system modules that are:

- as independent of each other as possible (low coupling),
- small in size so that there is no difficulty in understanding the logic and
- there is a high degree of component interaction within a module (high cohesion).

Consequently, an individual module can be designed, coded, tested and amended without too much reference to other modules of a design.

Because the overall design methodology includes identify and assessing hazards (expected error situations and solutions to expected error situations), the fact that there is high cohesion in modules, low coupling between modules and principals of information hiding (where data is encapsulated) ensures that a program can be employed in safety critical situations. For errors can be easily discovered, during the testing stages, and modifications to the offending parts of a design can easily be made without affecting the whole structure of a design.

Hence, a designer and user can have confidence in the resultant solution when a JSP methodology is used.

4.2. HARDWARE SPECIFICATIONS

4.2.1 The Temperature Monitoring Device (TMD)

The hardware of the temperature monitoring device must:

- be small and light enough to be fixed to a persons body and yet be comfortable .
- use very little power
- be able to have its batteries changed without losing its functions or data.
- be able to receive temperature sensor readings.
- be able to detect 0.1 degrees Celsius changes in temperature.
- be able to signal condition the temperature sensor readings so that they are in a range suitable for the ADC subsystem.
- be able to measure temperatures in the range 35.0 to 43.0 degrees Celsius.
- be able to convert the analog temperature readings into 8-bit digital values.
- be able to store 4K bytes worth of data (8-bit temperature readings).
- be able to receive push button signals that invoke the functions of the device.
- be able to receive serially date, time and start logging information, at a fixed baud rate.
- be able to transmit serially the data from a test, at a fixed baud rate.
- be able to illuminate individual light emitting diodes, on demand, that show the device's status.

4.2.2 The Data Transfer Device (DTD)

The hardware of the data transfer device must:

- be portable.
- be able to be powered by batteries
- be able to maintain the date and time of day, even when the device is switched off
- be able to display menu choices using alpha-numeric characters (on a 4 - line by 20 - character liquid crystal display).
- be able to input commands (to select menu choices) and input a new date time values from a hexadecimal keyboard.
- be able to receive 4K bytes of data serially, at a fixed baud rate.
- be able to store 4K bytes worth of data.
- be able to retransmit the 4K bytes of data serially, at a fixed baud rate.
- be able to reset the data transfer device via a push button.

4.2.3 The data Processing System (DPS)

- The data processing system must have an IBM PC system with:
- a copy of MS-DOS version 3.2 or greater as the operating system.
- a hard disk unit or a floppy disk unit to hold the files of data and the program.
- a serial communications port.
- a parallel printer port.
- an EGA/VGA graphics adaptor.
- an EGA/VGA colour monitor.
- a dot matrix printer capable of producing a hard copy output with a resolution of 360 dots per inch or greater.

4.3. THE SYSTEM MODEL

Sommerville (1987) states that, "once an initial analysis of the user's needs has been carried out, the next step is to produce a conceptual model of the software system. The conceptual model is a very high-level view of the system in which the major user services are identified , and their relationships documented".

The functions of the temperature monitoring system are shared between the hardware sub-systems and the software routines. Figure 11 shows the five main hardware components of the system, the two human participants, the data flow, and the originator of the commands which activate the software.

The system model diagram shows the following logical connections: (see Figure 11)

- the flow of data between the TMD, DTD and DPS,
- the origin for the input of commands, and
- the recipient of the results from the data processing system.

A high-level description of the complete temperature monitoring system is described below.

(i) The relationship between a person under test and three main parts of the hardware of the system is as follows:

The sensor is attached to a person's body for a period of up to 4 weeks (24 hours a day). The sensor sends a continuous temperature sensitive signal to the temperature monitoring system. At the start of a test the person under test connects the data transfer device to the temperature monitoring device. The data transfer device is commanded to download the date, time and a start signal to the temperature monitoring device. Immediately after the two devices are disconnected, the temperature monitoring device measures and stores temperatures at the rate of one reading every ten minutes. At any instance, before or during a test, the status of the device can be viewed. Also during a logging session significant times of events can be recorded by the TMD by pressing the appropriate push button. At the end of a test the two devices are reattached and the temperature monitoring device is commanded to download its stock of temperature measurements to either: the data transfer device or the data processing system. The two devices are then disconnected.

(ii) The relationship between a researcher and three main parts of the hardware of the system is as follows:

The researcher connects the data transfer device (or in some cases the temperature monitoring device) to the data processing system. The data processing device is commanded to upload the data from the test and store it in the PC's memory. The PC program processes the data and displays graphically the results from the test, on the VDU and as a hard copy on printer paper.

4.4. FUNCTIONAL REQUIREMENTS SPECIFICATIONS

Cooling (1991, p. 75) stated that

the functional requirements specifications relates to system behaviour. They describe: what the system does, when it does it and how it responds to deviations to the normal behaviour. They describe it's processes, the inputs to each process, the outputs, expected error situations and the solutions to these errors. Note, the requirement specifications should not define how these requirements are to be satisfied.

4.4.1. The Temperature Monitoring Device

The temperature monitoring device must perform the following tasks:

4.4.1.1. PROCESSES:

The processes are as follows:

- The device must have a signal conditioning unit that will convert the sensor signal into an analog voltage suitable for the range of values expected by the analog to digital converter (ADC). It also must output signals to the ADC interface, two reference voltages (voltage reference high (Vrh) and voltage reference low (Vrl)). These two reference voltages set the limits for the expected range of values to the ADC.

- A reset situation must initialise all the following device interfaces:

- initialise the ADC sub-system
- initialise the SCI sub-system.
- initialise the parallel ports.
- initialise the counter/timer sub-system.

-A reset situation must also perform all of the following:

- initialise the control program variables.
- clear the data buffer.
- wait for a command in the `ready' program mode.

- A command input from push buttons must invoke one of the following:

- **a device reset routine.**
- **the receipt of the date, time and a start monitoring signal from the serial port.**
- **the hardware to show the device's status.**
- **the uploading of: the date and time, and a start signal from the data transfer device.**
- the downloading of the data via the serial port to the data transfer device or to the data processing system.

4.4.1.2. INPUTS:

The inputs to the temperature monitoring device are as follows:

Command signals from the push buttons.
Serial information (date, time and a start command) at a fixed baud rate from the data transfer device.
An analog signal from the sensor that represents temperatures in the range: 35.0 to 43.0 degrees Celsius.

4.4.1.3. OUTPUTS:

The outputs from the temperature monitoring device are as follows:

Serial information consisting of: the data gathered from a test, the date and time of the start of the test, and the monitor device's identification label, to the data transfer device or to the data processing system (at a fixed baudrate)

The status of the temperature monitoring device on demand. The status must indicate:

- the condition of the battery.
- if the data memory is full/not full.
- if the device is in a ready state.
- if the device is monitoring data.
- if the device is transferring data.

4.4.1.4. EXPECTED ERROR SITUATIONS:

The expected error conditions caused by user mistakes and missing data are as follows:

More than one push-button pressed.

No start of block marker detected, date or time values in the wrong format, or no end of block marker detected; within 60 seconds of a upload command.

4.4.1.5. SOLUTIONS TO EXPECTED ERRORS:

The solutions to the aforementioned expected errors are as follows:

Acknowledge only the highest priority push-button input; clear the rest.
Reset the system and wait for a new command.

NB. The device must be able to read the data from the ADC, store the data, at the rate of one reading every 10 minutes for up to a period of 4 weeks. Then, when commanded, output the device's identification, date and time of the start of the test, and the data recorded during the current test.

NB. A data monitoring session is ended by either:

the data memory being full.

the data transfer command being detected from a push-button.

4.4.2. The Data Transfer Device

The data transfer device must perform the following tasks:

4.4.2.1. PROCESSES:

The data transfer device must perform the following processes:

The device must have a means of resetting itself, when commanded from a push button signal, and hence, perform the following tasks:

- initialise the SCI sub-system.
- initialise the parallel ports.
- initialise the hexadecimal keypad interface.
- initialise the control program variables.
- clear the data buffer.
- display the menu of commands.
- wait for a command from the hexadecimal keypad.

The data transfer device must be capable of displaying the system commands and allowing an input to invoke one of the following:

- force the device to reset the system and initialise the device.
- to display the current date and time of day (allow changes if required).
- to allow the down loading of : the date and time, and a start data logging signal to the temperature monitoring device.
- to allow the uploading of data from the data monitoring device.
- to allow the down loading of the data to the data processing device.
- to display the status of the device including the amount of data stored in memory.
- to display the data stored in memory.
- to the clearing of the previous data from the data buffer.

NB. The device must be able to store 4K bytes of data in a semi-permanent memory.

4.4.2.2. INPUTS:

The inputs to the data transfer device are as follows:

Commands from the hexadecimal keypad.

- The current date and time, from the hexadecimal keypad, to update the real-time clock chip.
- The temperature monitor device's identification, start of a test time and the data recorded during the test, from the serial communications port.

4.4.2.3. OUTPUTS:

The outputs from the data transfer device are as follows:

Menu commands on a display.

- A device's identification, date and time of a test, and the data from the test onto a display.
- The status of the device on a display.
- The serial information consisting of: the current date and time, and a start data logging signal are sent to the temperature monitoring device (at a fixed baud rate).
- The serial information consisting of:: the data gathered from a test, the recorded event times, the date and time of the start of the test, and the temperature monitoring device's identification label, are sent to the data processing system (at a fixed baud rate).
- Error messages to the display.

4.4.2.4. EXPECTED ERROR SITUATIONS:

The expected user errors and errors due to loss of data are as follows:
- A non-system command entered via the hexadecimal keypad.
- The date or time information input from the hexadecimal keypad is in the wrong format.
- No start of block marker detected, or no end of block marker detected; within 180 seconds of an upload command.
- Any other noticeable error.

4.4.2.5. SOLUTIONS TO EXPECTED ERRORS:

The solutions to the aforementioned expected errors are as follows:
- Ignore non-system commands from the hexadecimal keypad.
- Output an error message and display the expected format for the date and time.
- Output an error message after reading 4K bytes of data or after the time-out period.
- Reset the system via a command, or if a continuing error situation occurs press the reset push-button.

4.4.3. The Data Processing System.

The software for the processing system is designed to run on an IBM PC with the specifications described previously. The system must support the following functions:

4.4.3.1. PROCESSES:

The software is designed to run the following processes:

The PC system must: initialise the serial communications port and the parallel printer port and display on the monitor a menu of commands that are available to the data processing system. Then allow command choices to be entered via the PC keyboard to invoke:

- the viewing of a directory of files from a specified drive.
- the input of the data from the data transfer device, or from the temperature monitoring device, via a serial communication port (at a fixed baud rate).
- the saving of the current data in memory into a specified new data file, and store it in the secondary storage.
- the viewing of the raw data from either: the current test or from a file containing data from a previous test.
- the plotting of the results of the current test; on the VDU screen, and produce a hard copy, in graphical form, on the printer paper.

4.4.3.2. INPUTS:

The data processing system requires the following inputs:

- The commands from the PC keyboard.
- The data from a test via the serial communications port (at a fixed baud rate), originates from either:
  the data transfer device, or
  from the temperature monitoring device.

4.4.3.3. OUTPUTS:

The data processing system will produce the following outputs:

- The menu of commands on the VDU screen.
- A directory of files from a specified disk directory.
- The raw data from the current test or from a previous test.
- The processed data in the form of a graph; temperature versus time.
- A hard copy of the processed data in the form of a graph; temperature versus time.

4.4.3.4. EXPECTED ERROR SITUATIONS:

The data processing must be designed to cope with the following expected error situations:

- A non-system command is detected.

- No start of block marker detected, or no end of block marker detected; within 180 seconds of an upload command.

- A printer not ready error is detected.

- A directory or file not found error is detected.

4.4.3.5. SOLUTIONS TO EXPECTED ERRORS:

The data processing system will have the following solutions to the aforementioned expected error situations:

- Ignore non-expected commands from the PC keyboard.

- Display an error message if no start of block marker is detected within 180 seconds of receiving an uploading command or after reading 4K bytes of data.

- Display a printer not ready error message.

- Display a file not found error message.

4.5. DATA TYPES REQUIREMENT

This section of the `Software Requirement Document' defines the following data types:

(i) microcontroller I/O registers,
(ii) program parameters (constant values for the program),
(iii) variables used by the various routines of each program,
(iv) memory buffers required to store data or lookup tables,
(v) initial values used when the devices or processing system are reset.

4.5.1. Temperature Monitoring Device

4.5.1.1. microcontroller I/O registers.

These are Motorola defined names and addresses found in the M68HC11 Reference manual (M68HC11RM/AD).

TIMER SUB SYSTEM:

```
TMSK2     EQU  $1024 ; = $00  for no timer interrupts or no timer
                              scaling.

TFLG2     EQU  $1025          ; = $80  (-ve) when a TOF occurs.

TCNT      EQU  $100E ; 16-bit free-running counter. HB=100E
                                                  LB=100F
```

ADC SUB SYSTEM:

```
OPTION    EQU  $1039 ; The OPTION register is used to initialise
                       the ADC.
                     ; AND with #$BF  for CSEL = 0
                     ;  OR  with #$80  for ADPU = 1
ADCTL     EQU  $1030 ; = $10  multi channel channel ADR1 to
                       ADR3.
ADR1      EQU  $1031 ; = digital value (current temperature
                  reading).
ADR2      EQU  $1032 ; = digital value (current battery condition)
```

INPUT COMMANDS PINS AND OUTPUT STATUS PINS:

```
PACTL     EQU  $1026 ; DDRA3 = 0  to make PA3 an input,
                     ; I4/OC5  = 1  to enable the IC4 pin,
                     ; DDRA7 = 1  to make PA7 an output.
```

```
PORTA       EQU   $1000
                              ; PA0   i/p IC3  start uploading command
                              ; PA1   i/p IC2  start down loading command
                              ; PA2   i/p IC1  note an event command
                              ; PA3   i/p IC4  display device's status
                                               command
                              ; PA4   o/p OC4 monitoring data indicator
                              ; PA5   o/p OC3 memory full indicator
                              ; PA6   o/p OC2 battery condition low
                                                indicator
                              ; PA7   o/p OC1 transferring serial data
                                                indicator
TCTL1 EQU   $1020  ; = $00  to avoid output compare actions taking place

TMSK1       EQU   $1022  ; =$00  to disable i/p capture & o/p comp.
                           interrupts.

TFLG1 EQU   $1023  ; OC1F OC2F OC3F OC4F IC4F IC1F IC2F IC3F
                   ; read to detect flags being set, write a 1 to clear flag.
```

SCI SUB SYSTEM:

```
BAUD        EQU   $102B  ; set the baud rate
                         ; TCLR   0  SCP1 SCP2 RCKB SCR2 SCR1
SCR0
                         ;   0    0   1    1    0    0    1
1
                         ; 1200 BAUD = #$33

SCCR1 EQU   $102C  ; set M = 0 for 8 data bits

SCCR2 EQU   $102D  ; AND  with   #$0F   to disable SCI interrupts
                   ; OR   with   #$08   to enable the transmitter
                   ; OR   with   #$04   to enable the receiver

SCSR        EQU   $102E  ; the SCI status register.
                         ; if TDRE = 1    data transmitted,
                         ; (write to SCDR to clear TDRE flag).
                         ; if RDRF = 1    data received,
                         ; (read from SCDR to clear RDRF flag).

SCDR        EQU   $102F  ; the data register for serial receive/transmit
                           data.
```

PARALLEL PORTS:

```
PORTA    equ   $1000 ; an 8-bit I/O port,(for the commands and
                       indicators)

PORTB    EQU   $1004 ; an 8-bit output port, (expanded mode
                       memory)

PORTC    EQU   $1003 ; an 8-bit I/O port,    (expanded mode
                       memory)

PORTD    EQU   $1008 ; a  6-bit I/O port,    (SCI sub system)

DDRD     EQU   $1009 ; =$00 for inputs, note, SCI overrides i/ps

PORTE    EQU   $100A ; 8-bit input port,     (ADC sub system)
```

4.5.1.2. Parameters, Variables, Buffers and Initial Values. (user defined)

INITIALISATION:

```
IDENT 2 bytes ; 16-bit temperature monitoring device identification value

DATEBUF    14 bytes     ; ASCII buffer for date,time and end of block
                          character
DATEMAX    1 byte       ; end of date and time (ASCII) buffer

DAY        1 byte       ; start of test date     (BCD)
MONTH      1 byte       ;                        (BCD)
YEAR       1 byte       ;                        (BCD)
HOUR       1 byte       ;start of test time      (BCD)
MINUTE     1 byte       ;                        (BCD)
SECOND     1 byte       ;                        (BCD)

BUFFER     4K bytes     ; temperature readings data area
BUFFPTR    2 bytes      ; pointer into the 4K byte BUFFER
BUFFMAX    2 bytes      ; end of buffer limit

LOOKUP     256 bytes    ; temperature lookup table

EVENTB     256 bytes   ; 128 possible events can be recorded, each 16-
                        bit
                        ; value = the current AMOUNT
EVENTPTR   2 bytes      ; initially = #$0000 (inc by 2 each event)
```

TIMER SUB SYSTEM:

TENMIN	2 bytes	; initially = $0000 incremened for each TOF	
TILIMIT	2 bytes	; = #$4785	no. of TOFs in 10 mins
ONEMIN	2 bytes	; initially = $0000 incremented for each TOF	
TIMEOUT	2 bytes	; = #$0207	no. of TOFs in 1 minute

ADC SUB SYSTEM:

ADCDELAY	1 byte	; 100 uSec delay constant for powerup of ADC
TEMP11 byte		; digitised temperature reading
TEMP21 byte		; temp value from LOOKUP table
		; 4-bits represent values 34 to 43
		; 4 bits represent values .0 to .9
TEMP31 byte		; spare byte
AMOUNT	2 bytes	; no. of temperature readings recorded
BATLOW	1 byte	; battery condition; low level value

INPUT CAPTURE:

UPLOAD	1 byte	; if '1'	an upload in progress
DNLOAD	1 byte	; if '1'	a down load in progress
EVENT	1 byte	; if '1'	an event noted
DISPSTAT	1 byte	; if '1'	a display status in progress

OUTPUT STATUS:

MONITOR	1 byte	; set to #$FF if monitoring data
MEMFULL	1 byte	; set to #$FF if memory is full (end of test)
BATTERY	1 byte	; set to #$FF if battery condition is low
TRANSFER	1 byte	; set to #$FF if a upload/ download in progress
EVENTFUL	1 byte	; set to #$FF if event buffer full
READFLAG	1 byte	; set to #$FF for every 10 minute timeout

SCI SUB SYSTEM:

PTR2BUF	2 bytes	; = BUFFER initially	(inc. for every data uploaded)
PTRTMAX	2 bytes	; = BUFFPTR + AMOUNT	(when uploading)
PTREVENT	2 bytes	; = EVENTB	(inc. for every event uploaded)
PTREMAX	2 bytes	; = EVENTB + EVENTNO	(when uploading)
BEGCODE	1 byte	; = #$2A	start of serial block marker (*)
ENDCODE	1 byte	; = #$23	end of serial block marker (#)

4.5.2. Data Transfer Device

4.5.2.1.microcontroller I/O registers.

These are Motorola defined names and addresses found in the M68HC11 Reference manual (M68HC11RM/AD).

TIMER SUB SYSTEM:

```
TMSK2       EQU   $1024  ; = $00   for no timer interrupts or no timer
                            scaling.

TFLG2       EQU   $1025  ; = $80   (-ve) when a TOF occurs.

TCNT        EQU   $100E  ; 16-bit free-running counter. HB=100E
                            LB=100F
```

HEXADECIMAL KEYPAD

```
PORTA       EQU   $1000
                         ; PA0  i/p     A-0     keypad strobe

PORTEEQU    $100A

                         ; PE7  i/p     E-7     keypad data     (bit-3)
                         ; PE6  i/p     E-6     keypad data     (bit-2)
                         ; PE5  i/p     E-5     keypad data     (bit-1)
                         ; PE4  i/p     E-4     keypad data     (bit-0)
```

LIQUID CRYSTAL DISPLAY

```
PORTA       EQU   $1000
                         ; PA4  o/p     A-4 LCD control signal  (RS)
                         ; PA5  o/p     A-5 LCD control signal  (R/W)
                         ; PA6  o/p     A-6 LCD control signal  (E)
```

REAL-TIME CLOCK CHIP

```
PORTD       EQU   $1008

                         ; PD2  i/p     D-2     MISO   receive data
                         ; PD3  i/p     D-3     MOSI   transmit data
                         ; PD4  i/p     D-4     SCK    clock signal
                         ; PD5  i/p     D-5     SS     slave select
DDRD        EQU   $1009

            ; D7    D6    D5    D4    D3    D2    D1    D0
            ; 1     1     0     1     1     0     1     0
```

```
SPCR        EQU   $1028

            ; D7   D6    D5    D4    D3    D2    D1    D0
            ; 0    1     0     1     0     0     1     1

            ; D0   clock rates     (divide by 32)
            ; D1
            ; D2   clock phase
            ; D3   clock normally low when not transmitting
            ; D4   master mode
            ; D5   normal CMOS outputs
            ; D6   SPI sub system 'ON'
            ; D7   disable SPI interrupts

SPSR        EQU   $1029

            ; D7   SPIF            transfer complete flag
            ; D6   WCOL            write collision error
            ; D4   MODF            mode error

SPDR        EQU   $102A

            ; the serial peripheral data register
```

SCI SUB SYSTEM:

```
BAUD        EQU   $102B  ; set the baud rate
                         ; TCLR  0  SCP1 SCP2 RCKB SCR2 SCR1
SCR0
                         ;   0    0   1    1    0    0    1    1
                         ; 1200 BAUD = #$33

SCCR1 EQU   $102C  ; set M = 0 for 8 data bits

SCCR2 EQU   $102D  ; AND  with   #$0F   to disable SCI interrupts
                   ; OR   with   #$08   to enable the transmitter
                   ; OR   with   #$04   to enable the receiver

SCSR        EQU   $102E  ; the SCI status register.
                         ; if TDRE = 1   data transmitted,
                         ; (write to SCDR to clear TDRE flag).
                         ; if RDRF = 1   data received,
                         ; (read from SCDR to clear RDRF flag).

SCDR        EQU   $102F  ;the data register for serial rec/trans data.
```

PARALLEL PORTS:

```
PORTA       EQU   $1000  ; an 8-bit I/O port,  (for the keypad and LCD)

PORTB       EQU   $1004  ; an 8-bit output port,       (expanded mode
memory)

PORTC       EQU   $1003  ; an 8-bit I/O port,          (expanded mode
memory)

PORTD       EQU   $1008  ; a  6-bit I/O port,   (SCI, SPI and LCD)

DDRD        EQU   $1009  ; =$DA ,              (for the SCI, SPI and
LCD)

PORTEEQU    $100A  ; 8-bit input port,         (hexadecimal keypad data)
```

4.5.2.2. Parameters, Variables, Buffers and Initial Values. (user defined)

INITIALISATION:

```
IDENT 2 bytes                    ; 16-bit TMD identification value

DAY         1 byte               ; start of test date      (BCD)
MONTH       1 byte               ;                         (BCD)
YEAR        1 byte               ;                         (BCD)
HOUR        1 byte               ;start of test time       (BCD)
MINUTE      1 byte               ;                         (BCD)
SECOND      1 byte               ;                         (BCD)

BUFFER      4K bytes             ; temperature readings data area
BUFFPTR     2 bytes              ; pointer into the 4K byte BUFFER
BUFFMAX     2 bytes              ; end of buffer limit

EVENTB      256 bytes          ; 128 possible events can be recorded, each 16-
                                 bit
                                 ; value = the current AMOUNT
EVENTNO     2 bytes            ; initially = #$0000   (increnented by each event)

PTREVENT    2 bytes              ; = EVENTB     (inc. for every event uploaded)
PTREMAX     2 bytes              ; = EVENTB + EVENTNO  (when uploading)

BEGCODE     1 byte               ; = #$2A        start of serial block marker  (*)
ENDCODE     1 byte               ; = #$23        end of serial block marker  (#)
```

TIMER SUB SYSTEM:

ONEMIN	2 bytes	; initially = $0000	incremened by a TOF
TILIMIT	2 bytes	; = #$0727	no. of TOFs in 1 minute

HEXADECIMAL KEYPAD:

KEYBUF	10 bytes	; initially = $00 00 00 00 00 00 00 00 00 00
KEYNUM	1 byte	; number of keypad entries

LCD MODULE:

TEMP	4 bytes	; tens
		; units
		; decimal point
		; tenths
TIMES	8 bytes	; delay times during initialisation
		; $ 30 10 01 01 01 10 30 01
INSTRUCT	8 bytes	; instructions used to initialise the LCD module
		; $ 30 30 30 20 20 08 01 0F
MSG1	20 bytes	
MSG2	20 bytes	
MSG3	20 bytes	
MSG4	20 bytes	

REAL-TIME CLOCK CHIP

DAY	1 byte	; start of test date	(BCD)
MONTH	1 byte	;	(BCD)
YEAR	1 byte	;	(BCD)
HOUR	1 byte	;start of test time	(BCD)
MINUTE	1 byte	;	(BCD)
SECOND	1 byte	;	(BCD)

4.5.3. Data Processing System

4.5.3.1 Program control variables (user defined)

```
ERRCODE    DB    0                        ; type of error

HANDLE     DW    0                        ;  a handle to an opened file

PATH       DB    64   DUP (00h)                ; file specification

DTA        DB    64   DUP (00h)                ; Data Transfer Area

IDENTV     DB    5Ah , 0a5h               ; TMD identification

                                          ; date and time values

DATEBUF DB    31h, 33h, 30h, 37h, 39h, 33h, 30h, 39h, 34h, 35h, 30h,
30h

ROW        DB    1                         ; row on screen
COL        DB    1                         ; column on screen
```

4.5.3.2. Buffer Space for Data and Events (user defined)

```
BUFFER     DB    4096    DUP (00h)    ; buffer area  4K bytes of
data

EVENTB     DB     256    DUP (00h)    ; buffer area for 128 words
                                      ; event times
```

4.5.3.3. VDU screen error messages (user defined)

```
MSG1 DB    "Serial port not initialised."

MSG2 DB    "Printer port not initialised."

MSG3 DB    "Type `C' to return to the main MENU screen."

MSG4 DB    "  ****  LOADING  FILE  CONTENTS  ERROR  ****"
```

4.5.3.4. VDU screen menu messages (user defined)

MSGMENU

```
Type   `E'   to  EXIT  program; back to DOS.

Type   `I'   to upload data from the serial port.

Type   `D'   to view a directory of files.

Type   `R'   to view the raw date from memory.

Type   `L'   to load the raw data from a file.

Type   `S'   to save the raw data to   a file.

Type   `P'   to view the processed data.
```

MSGINPUT

```
UPLOADING  RAW  DATA  FROM   TMD or DTD

*****************************************

IDENTIFICATION  OF  THE TMD        =

STARTING  DATE  OF TEST            =
```

MSGRAW

```
DISPLAYING  RAW  DATA  FROM  MEMORY"
***************************************

IDENTIFICATION  OF  THE TMD        =


STARTING  DATE  OF TEST            =
```

MSGDIR

```
DISPLAYING  A  DIRECTORY  OF  FILES
***********************************


Input the complete path of the directory.
For example,    A:\test\*.dat
    or           C:*.*
```

MSGLOAD

```
LOADING DATA FROM A SPECIFIED FILE
****************************************
Input the complete path and filename.

For example,   A:\test\test.dat
    or         C:trial.dat
```

MSGSAVE

```
SAVING DATA TO A SPECIFIED FILE
************************************

Input the complete path and filename.

For example,   A:\test\test3.dat
    or         C:trial56.dat
```

4.6. NON-FUNCTIONAL REQUIREMENTS

Cooling (1991, p. 76) states that "the non-functional system requirements specifications should define":

- How well a function should be performed (PERFORMANCE).
- How the system connects to its environment (INTERFACES).
- What limitations are placed on the design (DESIGN CONSTRAINTS).
- Anything that does not fit into the other groups (OTHER CONSTRAINTS).

Hence, the three main parts of the temperature monitoring system, wh h require software, are described as follows:

4.6.1. The Temperature Monitoring Device

4.6.1.1. PERFORMANCE:

Computational time:

The main task of the temperature monitoring device is to record temperature measurements. This is required at the rate of one reading every 10 minutes.

The next most important tasks are to down load and upload serial data; this is to be done at a 1200 baud rate during the testing of the prototype (but could be at any suitable baud rate).

The displaying of status information is to be activated as long as the push-button is pressed. This function should not interrupt the actual reading of a temperature measurement, but can interrupt the down loading or uploading of serial data.

Storage capacity:

The device requires the following memory space:

512 bytes of RAM	for program stack and variables,
8K bytes of ROM	for the control program,
4K bytes of EEPROM	for all the temperature readings,
512 bytes of EEPROM	for the program parameters: identification of device, date and time.

4.6.1.2. INTERFACES:

(a) Analog input signals:

There is one analog input signal; the temperature signal from the signal conditioning unit. This signal is accompanied by two reference voltages (Vrl) and (Vrh) which define the upper and lower limits of the temperature signal. The three signals are input into the ADC sub system.

The 8-bit ADC has a total error of + or - 1 LSB. Each conversion is accomplished in 32 microcontroller unit (MCU) E clock cycles. Note, the MCU operates at
8 M Hz, therefore (E = 0.125 micro Seconds). Hence, each conversion takes
4 micro Seconds.

The ADC sub system, for one analog input, uses the following I/O registers:

$1030	ADCTL	A/D Control Register
$1031	ADR1	A/D Result 1

(b) Serial communication signals:

The uploading and down loading takes place via the system's Serial Communications Interface (SCI) sub system, at a 1200 baud rate.

The SCI is a full-duplex asynchronous interface with a standard NRZ format (one start bit, 8 data bits, and one stop bit) with a variety of programmable baud rates.

The SCI sub system uses the following I/O registers:

$1008	PORT D	I/O port D data register
$102B	BAUD	SCI baud rate register
$102C	SCCR1	SCI control register 1
$102D	SCCR2	SCI control register 2
$102E	SCSR	SCI status register
$102F	SCDR	SCI data register

(c) Timer/Counter register values:

The timing of events in the temperature monitoring system is achieved using the microcontroller's 16-bit free-running counter, and the timer/counter interrupt flag registers 1 and 2.

Note, when the count changes from $FFFF to $0000, the timer overflow flag (TOF) bit is set in the timer interrupt flag register 2 (TFLG2).

The free-running counter is driven by the MCU E clock. Hence, each count is incremented every 0.500 micro seconds; each TOF bit is set every 32.77 milli seconds.

The Timer sub system is also used to detect push-button presses (input capture) and for diplaying the status of the device (output compare).

The register that are required for this device, from the Timer sub system, are as follows :

$100E	TCNT	Timer counter register high byte
$100F		" " " low byte
$1020	TCL1	Timer control register 1
$1021	TCL2	Timer control register 2
$1022	TMSK1	Timer interrupt mask register 1
$1023	TFLG1	Timer interrupt flag register 1
$1024	TMSK2	Timer interrupt mask register 2
$1025	TFLG2	Timer interrupt flag register 2

(d) Parallel ports
Although there are no parallel data transfers between the system components, the following microcontroller ports are utilised as follows:

PORT A: This port has to be configured for: 4 input capture pins, and 4 output compare pins. Port A is to be used as follows:

pin no.	direction	name	description
PA0	i/p	IC3	start uploading
PA1	i/p	IC2	start down loading
PA2	i/p	IC1	note an event
PA3	i/p	IC4	display the device status
PA4	o/p	OC4	monitoring data
PA5	o/p	OC3	(memory full) / (memory not full)
PA6	o/p	OC2	battery voltage is alright
PA7	o/p	OC1	transferring data

Note, pin PA7 is configured as an output by setting bit-7 =`1' (DDRA7) of the pulse accumulator control register (PACTL).

Press push-buttons and show device status functions use the following I/O registers:

$1000	PORTA	I/O port A
$1026	PACTL	Pulse accumulator control register

PORT B: This port is used during the external mode of operation for the upper 8 bits of an external memory address.

PORT C: This port is used during the external mode of operation for the 8 bidirectional data lines and the low byte of an external memory address.

PORT D: This port does not need to be configured via the I/O registers to make pins PD0 act as an input (RxD) and PD1 act as an output (TxD). The SCI subsystem takes control of this port when it is required for a serial data transfer.

pin no.	direction	name	description
PD0	i/p	RxD	receive data
PD1	o/p	TxD	transmit data

The ADC function uses the following I/O register to transfer the serial data.

$1008	PORTD	I/O port D

PORT E: This port is always configured as an input port by the microcontroller itself. For this device only pin PE0 is used as an input port for the analog signal (the temperature signal).

$100A	PORTE	input port E

4.6.1.3. DESIGN CONSTRAINTS:

Programming language:
M6800 family assembly language for the Portable Cross Assembler (PASM) must be used.
Processor type:
MC68HC11 microcontroller.
Maximum memory capacity:
64K bytes of primary memory space.

4.6.1.4. OTHER CONSTRAINTS:

Maximum physical size:
6 cm x 5 cm x 3 cm
Maximum weight:
120 grams
Temperature operating range:
- 10 degrees Celsius to + 45 degrees Celsius
Safety and Comfort:
This device has to be attached to a patient under test so it must be comfortable against the patient's body. It must not have any sharp edges. It could be designed to be worn on the upper arm of a patient and therefore have a strap attached to it.
The device must be designed to operate from low voltage batteries; hence safe from electrical shocks.

Shock resistant:
The device must be able to absorb the shock from being dropped from a small height; say, 6 feet.

4.6.2. The Data Transfer Device

4.6.2.1. PERFORMANCE:

Computational time:

The four main tasks of the data transfer device are as follows:

(i) To maintain the time of day; this is done by communicating with a real-time clock chip (the MC68HC86T1). The date and time have to be read just prior to down-loading the date and time to the temperature monitoring device. Note, there are also commands that enable the user of the device, to read and change the date and time from the real-time clock chip, in order to maintain the correct date and time.

(ii) The date, time and a start test signal is down-loaded to the temperature monitoring device at the beginning of a monitoring session.

(iii) At the end of a temperature monitoring session the data is uploaded from the temperature monitoring device, and stored in memory waiting to be transferred to the data processing system.

(iv) Some time later the data is down-loaded to the data processing system.

NB, all the aforementioned uploading and down-loading is done serially at a 1200 baud rate

The secondary tasks performed by the data transfer device are as follows:

When any of the four main tasks are not being performed, a command menu should be displayed. The command menu, apart from showing the four main tasks, should display commands to allow the following tasks:

(v) The device to be initialised; this should only be necessary when unexpected, unsolvable, problems arise.
(vi) The status of the device to be displayed.
(vii) The data stored in the 4K byte buffer to be displayed.
(viii) The data buffer to cleared prior to an upload of more data from a temperature monitoring session.

Storage capacity:

The device requires the following memory space:

512 bytes of RAM	for program stack and variables,
8K bytes of ROM	for the control program,
4K bytes of EEPROM	for all the temperature readings,
512 bytes of EEPROM	for the program parameters: identification of device, date and time.

4.6.2.2. INTERFACES:

(a) Analog input signals:

There are no analog signals associated with the data transfer device.

(b) Serial communication signals:

The uploading and down loading takes place via the system's Serial Communications Interface (SCI) sub system, at a 1200 baud rate.

The SCI is a full-duplex asynchronous interface with a standard NRZ format (one start bit, 8 data bits, and one stop bit) with a variety of programmable baud rates.

The SCI sub system uses the following I/O registers:

\$1008	PORT D	I/O port D data register
\$102B	BAUD	SCI baud rate register
\$102C	SCCR1	SCI control register 1
\$102D	SCCR2	SCI control register 2
\$102E	SCSR	SCI status register
\$102F	SCDR	SCI data register

(c) Timer/Counter register values:

The timouts used for detecting expected errors during the transfer of data between the three main parts of the system are acheived using the microcontroller's 16-bit imer/counter sub system.

The free-running counter is driven by the MCU E clock. Hence, each count is incremented every 0.500 micro seconds; each TOF bit is set every 32.77 milli seconds.

The register that are available to this device, from the Timer sub system, are as follows:

\$100E	TCNT	Timer counter registerhigh byte
\$100F		" " " low byte
\$1020	TCL1	Timer control register 1
\$1021	TCL2	Timer control register 2
\$1022	TMSK1	Timer interrupt mask register 1
\$1023	TFLG1	Timer interrupt flag register1
\$1024	TMSK2	Timer interrupt mask register 2
\$1025	TFLG2	Timer interrupt flag register 2

(d) Parallel ports

PORT A: This port has to be configured for: 4 input pins, and 3 output pins. Port A is to be used as follows:

pin no.	direction	name	description	
PA0	i/p	IC3	a hexadecimal keypad strobe	
PA1	i/p	IC2	not used	
PA2	i/p	IC1	not used	
PA3	i/p	IC4	not used	
PA4	o/p	OC4	LCD control signal	(RS)
PA5	o/p	OC3	LCD control signal	(R/W)
PA6	o/p	OC2	LCD control signal	(E)
PA7	?	OC1	not used	

\$1000	PORTA	I/O port A
\$1026	PACTL	Pulse accumulator control register

PORT B: This port is used during the external mode of operation; for the upper 8 bits of an external memory address.

PORT C: This port is used during the external mode of operation; for the 8 bidirectional data lines, and the low byte of an external memory address.

PORT D: This port does not need to be configured via the I/O registers to make pins PD0 act as an input (RxD) and PD1 act as an output (TxD). The SCI subsystem takes control of this port when it is required for a serial data transfer.

Port D also acts as an interface for the real-time clock chip (M68HC68T1). Hence, this port needs to be configured via the DDRD I/O register to make pins PD2 to PD5 act for the SPI sub system as follows:

pin no.	direction	name	description
PD0	i/p	RxD	receive data
PD1	o/p	TxD	transmit data
PD2	i/p	MISO	RxD receive data
PD3	o/p	MOSI	TxD transmit data
PD4	o/p	SCK	serial clock signal
PD5	i/p	SS	slave select

The SCI sub system uses the following I/O register to transfer the serial data.

$1008	PORTD	I/O port D

PORT E: This port is always configured as an input port by the microcontroller itself. For this device pins PE4 through to PE7 are used as an input port for the hexadecimal keypad data (a value representing the key pressed)

pin no.	direction	name	description	
PE7	i/p	E-7	keystroke data	(bit-3).
PE6	i/p	E-6	keystroke data	(bit-2).
PE5	i/p	E-5	keystroke data	(bit-1).
PE4	i/p	E-4	keystroke data	(bit-0).

$100A	PORTE	input port E

4.6.2.3. DESIGN CONSTRAINTS:

Programming language:
M6800 family assembly language for the Portable Cross Assembler (PASM) must be used.

Processor type:
MC68HC11 microcontroller.

Maximum memory capacity:
64K bytes of primary memory space.

4.6.2.4. OTHER CONSTRAINTS:

Maximum physical size:
210 cm x 180 x cm x 6 cm

Maximum weight
750 grams

Temperature operating range:
- 10 degrees Celsius to + 45 degrees Celsius

Shock resistant:

The device must be able to absorb the shock from being dropped from a small height; say, 6 feet.

4.6.3. The Data Processing System

4.6.3.1. PERFORMANCE:

Computational time:

The two main tasks for the data processing system are as follows:

(i) Data is to be input via the serial communication port at the rate of 1200 baud.

(ii) The incoming data from a test must be processed to produce a `Time versus Temperature' plot, on the VDU screen, and onto a printer that is capable of plotting dot matrix graphics.

The secondary tasks for the data processing system are:

(iii) To save the raw data in a secondary storage file.

(iv) To view either the current raw data or raw data from a file.

(v) To view a MS-DOS directory of files.

NB. All five choices of tasks are from a menu of commands that are shown on the VDU. screen.

Storage capacity:

At least 500K bytes of RAM are required for the program, the data and for the operating system's use. Including 4K bytes of RAM for the storage of the raw data.

4.6.3.2. INTERFACES:

(a) Analog input signals:

none

(b) Serial communication signals:

The serial port has a standard RS232C interface. It has a full-duplex asynchronous interface with a variety of programmable baud rates.

The data received will have: one start bit, 8 data bits and one stop bit. The data will be transmitted at a 1200 baud rate.

(c) Parallel ports:

The standard IBM PC has a printer port with a parallel interface. The parallel interface consists of:

8 data lines	o/p
4 control lines	o/p
5 status lines	i/p

If direct control of the interface is required, then the following I/O addresses are needed:

output data	03BCh	or	0378h
output control	03BEh		037Ah
input status	03BDh		0379h

(d) Graphics interface:

The IBM PC is expected to have an EGA/VGA graphics adaptor with the following video mode:

Mode	Type	Resolution	Colours	Adaptor
16	graphics	640 x 350	16	EGA

(e) Software interfaces:

The application software will be designed to interface with MS-DOS and in particular with the BIOS routines when necessary.

The chosen high level language will interface with the BIOS routines that control the serial port and the parallel printer port.

If the operating system BIOS routines are required to produce the VDU graphics, then the following BIOS services are available:

set video mode	AH = 00
set the background colour	0B
set the colour palette	0B
set the palette registers	10
write a pixel dot	0C
read a pixel dot	0D
write a character and attribute	0E

4.6.3.3. DESIGN CONSTRAINTS:

Operating System:

MS-DOS version 3.2 or later.

Programming language:

A MASM assembly language including BIOS services and DOS functions is all that is necessary to write a simple menu driven program support program The BIOS services and DOS functions are used when dealing with all the I/O interfaces, otherwise the assembly language can easily cope with handling the bytes of data.

Processor type:

The Intel 80X86 family of microprocessors.

Maximum memory capacity:

1 mega byte of primary memory.

500 K bytes of RAM.

4.6.3.4. OTHER CONSTRAINTS:

none

4.7. STRUCTURED ANALYSIS

Introduction

Creating data flow diagrams (DFD) is one of the main stages of the JSP design methodology used in this thesis. The designer examines the specification from the software requirements specifications in order to produce a multi-level graphical representation of the system. The process of creating DFD and then transforming DFDs into Jackson structure diagrams is all part of structured analysis.

Pressman (1992) [page 207, "Software Engineering: A Practitioner's Approach"] states that structured analysis is a model building activity. The models depict information (data and control) flow and content. They depict the essence of what must be built.

Tom DeMarco (1979) [page 15, "Structured Analysis and System Specifications"] establishes the primary goals of an analysis method as follows:

- The products of analysis must be maintainable.
- Graphics have to be used whenever possible.
- There is a need to differentiate between logical and physical considerations.
- There is a need to keep track of and evaluate interfaces.

The basic notation of data flow diagrams DFDs

Pressman (1992) [page 208, "Software Engineering: A Practitioner's Approach"] informs us that information is transformed as it flows through a computer-based system. The system accepts input in a variety of forms, applies hardware, software and human elements to transform input into output, and then produces output in a variety of forms.

Structured analysis is an information flow and content modelling technique where:

One or more inputs are shown as arrows.

A single information transform is noted by a bubble.

Data that is to be stored for use by one or more processes are represented by two thick straight lines.

External entities are represented by boxes.

One or more outputs are shown as arrows.

It should be noted that the model may be applied to the entire system (level 0) or to the software elements only (levels 1, 2, 3, 4 etc.). The key is to represent information fed into and produced by a transform.

Data Flow-Oriented Design

Data Flow-Oriented Design can be described as a multi-step process in which representation of data structure, program structure and procedure are synthesised from information requirements. The design process is information driven.

A data flow-oriented method of design provides a systematic approach for the derivation of program structure. Beginning with a fundamental system model, information may be represented as a continuous flow that undergoes a series transformations as it evolves from input to output.

Modularity

Bell et al (1987) [page 27, "Software Engineering: A Programming Approach"] state that the essence of good modularity is to have components of a system as independent of each other as possible.

In programming, a module is any current or future mechanism for dividing software into manageable portions. A module should occupy no more than a page of information as it is difficult to understand logic that spills over from one page to another. Modules should also be made to be as clear as possible.

Modules should have the following characteristics:

- optimised size (one page or less),
- maximum cohesion,
- minimum coupling and
- information hiding.

Information Hiding

The principle of information hiding means that, at the end of a design process, any data is accessed only via certain, well defined, specific procedures or subprogrammes. It is a method of structuring a program in such a way that a piece of encapsulated data cannot be accessed directly.

Note, structured analysis should be performed in such a way that:

- Changes to design should be confined to as few modules as possible(preferably one).

- The software interfaces between modules should be as simple as possible and only be a means of calling subprograms rather than a means of accessing shared data.

- For the purpose of testing and maintenance of a program, it should be possible to understand individual modules independently of each other. The aim is to have clearer separation between modules .

Coupling and Cohesion

Coupling and cohesion are terminology and classification schemes for describing interactions between modules Bell et al (1987, p 34).

Software engineers are aiming at producing software modules with a minimum of interaction between them (low coupling) and conversely, a high degree of interaction within a module (high cohesion). Only then, an individual module can be designed, coded, tested and amended without referring to other modules.

The aim of software design is to have: weak coupling and strong cohesion within a program structure.

Coupling design criteria include:

- Modules should have only one entry point and only one exit point.
- As few parameters as possible should be passed between modules in a procedure call.
- Undesirable to have shared or global data.
- Accessing or modifying data within another module is undesirable.

The various types of cohesion that exist include:

- Coincidental cohesion, in which components are in a module by coincidence is undesirable.
- logical cohesion, in which a module performs a set of independent but logically similar functions should be avoided.
- Temporal cohesion, in which functions are related in time.
- Sequential cohesion, in which operations in a module collaborate to modify a piece of data are encouraged in a module.
- Functional cohesion, is employed in a module where operations contribute towards performing a well-defined task.

Transform Analysis

Pressman (1992) [page 208, "Software Engineering: A Practitioner's Approach"] states that transform analysis is a set of design steps that allows a DFD to be mapped into a template for program structure. The design steps are defined as follows:

step 1 Review the system model, the system requirements and the software requirements specifications in order to produce a level 0 and all level 1 data flow diagrams (DFD).

step 2 Review and refine data flow diagrams for the software. Information from the software requirements specifications is examined to produce DFDs that show greater detail (level 2, 3, 4 etc.). Lower level DFDs are produced until each module contains transforms with a high degree of cohesion. That is, each transform performs a single discrete function.

step3 Isolate the transform centre by specifying incoming and outgoing flow boundaries. Incoming flow is described as a path in which information is converted from external to internal form. Outgoing flow is when information is converted to external form. Then dotted lines may be drawn on DFDs to illustrate the input and output boundaries.

step 4 Perform first level factoring. This establishes graphically a program structure in which the top-levels show the overall control and decision making modules, the mid-levels modules perform some control and a moderate amount of work, whereas, the low-level modules perform most input, computational and output work.

step 5 Second level factoring. This involves mapping, on a one to one basis, the DFD processing components onto a Jackson program structure diagram. The individual transforms (bubbles) of a DFD are mapped onto a structure diagram, starting from the centre boundary and moving outwards. For example, a data flow diagram may be mapped onto a structure diagram that exhibits a main controller and three other components: input, processing and output.

step 6 Refine program structure (using design heuristic's) for improved software quality. A first-cut program structure can always be refined to have a structure that employs modules with good cohesion and low coupling. Hence, the program can be implemented without difficulty. The program can be tested without confusion and maintained without grief.

Design Heuristic's

Improvements to program structure can be made by applying the following guidelines:

- aim for modular independence,
- attempt to avoid situations with a high fan-out,
- make sure that all modules affected by decisions are at a lower level,
- evaluate module interfaces to reduce complexity,
- strive for single entry single exit modules and
- package software based on design constraints and portability requirements.

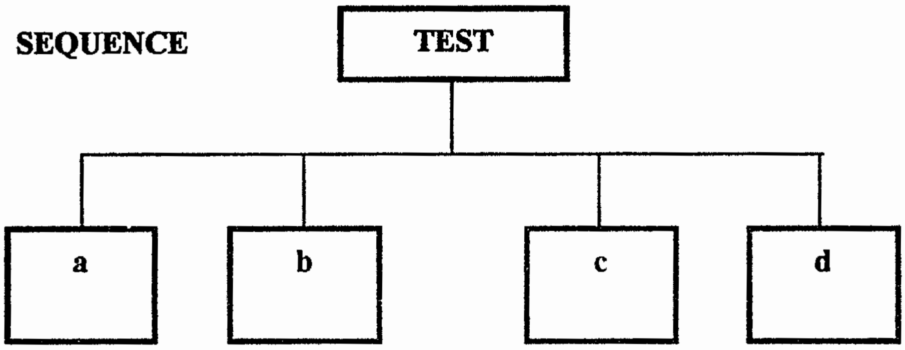
4.8. JACKSON STRUCTURED DIAGRAMS

Jackson structured diagrams have been used throughout the Temperature Monitoring System software documentation. Cooling (1991, p. 171) states that "Jackson structure diagrams can be used to show the structure of a program. They can also be used for language independant design. Jackson structured diagrams have three basic constructs which make them ideal for designing and documenting high level, medium level and assembly language programs". The constructs are:

- SEQUENCE,
- SELECTION and
- ITERATION.

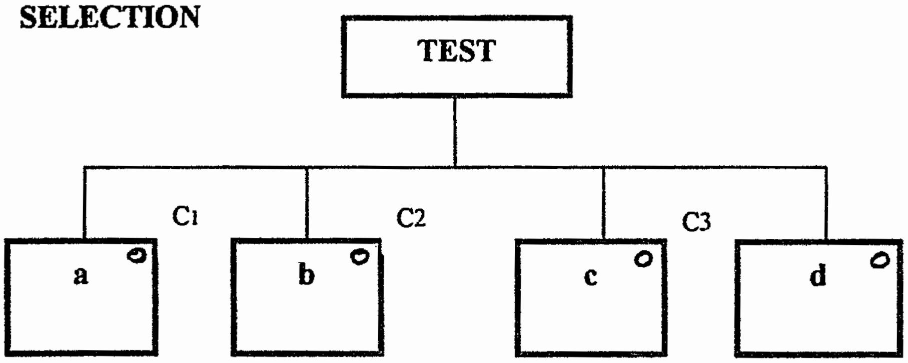
Consider a program routine called `TEST', which consists of 4 small modules (a, b, c and d). Each module could be either: labeled in-line-code, or a subroutine.

SEQUENCE



The structure diagram shown above implies that the module TEST passes control sequentially to a, b, c then d. Control is then passed back to TEST.

SELECTION



The diagram shown above implies that only one of the modules (a,b,c or d) will be executed. The small circles at the top right hand corner of each box indicates that the module needs to be selected before the functions relating to that module are executed. The symbols C1, C2, C3 and C4 are symbols representing the conditions that have to be met for the selection of that module. Control is always given back to TEST after a module's functions have been accessed.

For example, the conditions could be:

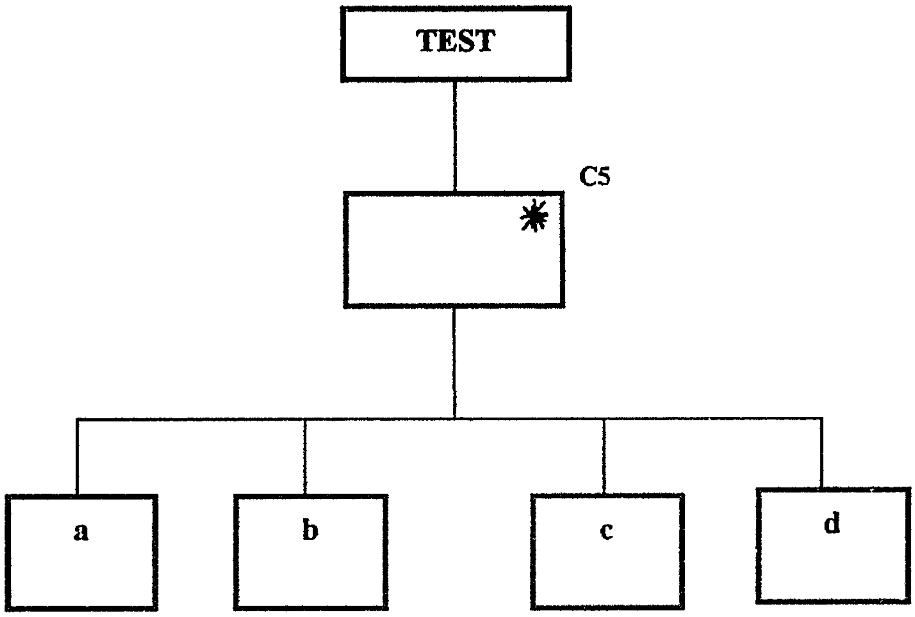
C1 `a' is selected when key 1 is pressed,

C2 `b' is selected when key 2 is pressed,

C3 `c' is selected when key 3 is pressed,

C4 `d' is selected when key 4 is pressed.

ITERATION The diagram shown below implies that the functions relating to the modules **a** through to **d** will be executed `n' times. The number of iterations will depend upon the condition set by C5. The asterisk (*) indicates that the components of a module will be executed in an iterative manner.



CONDITIONS

Modules associated with iteration or selection have their conditions listed in a table. For example,

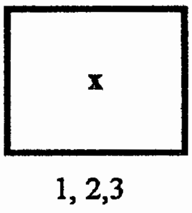
C1.	repeat 7 times
C2.	selected when key '1' pressed
C3.	selected when key '2' pressed

In the example shown above, the condition could be:

C5 the modules a,b,c and d will be sequentially executed 10 times.

FUNCTION NUMBERS

In Jackson structured design, each function that is defined within a program is allocated a function number. When a module is defined, the module is given a name, say x, and the function numbers are listed below the terminal module's icon, as shown below.



functions:

1. Move cursor to a specified position.
2. Input a character from the hexadecimal keypad.
3. Display the character on the LCD screen.

A function is defined once but can be called and used many times throughout a program. The function is referenced by its function number.

A module is defined once but can be called and used many times throughout a program. The module is referenced by its function number, which is specified in the list of functions. Note, the source code assembler has a restriction of up to eight unique characters for lables, variables and procedure names. Hence, the desired meaningful names cannot always be used, but the descriptions of the functions to be carried out by a module can easily be looked up from the lists of functions that are recorded in numerical order.

A Jackson structure diagram may consist of a hierarchy of constructs. Note, the rules are: a group of modules at any level of a diagram must be of the same type (no mixture of sibling types is not permitted). Also there can be no iteration siblings.

Modules connected together at the same level must be either:

. executed one after the other, from left to right or

. only one selected module is executed .

Functions within a terminal module must be defined in their order of execution; the function numbers will be displayed from left to right, in the order they will be executed. The definitions will be listed from top to bottom; as they would appear in a program listing.
Iterations are also from modules, shown left to right, and from functions within a module, defined from top to bottom.
Note that, the terminal modules shown in a hie[illegible]chy of a Jackson structure diagram may have their functions and conditions defined in the form of a list.

4.9 CHOICE OF PROGRAMMING LANGUAGE

Introduction

Programming can be defined as a process of converting system specification into useable machine code instructions to produce a desired result. Programming a computer to solve a problem involves two chores:

- The problem must be broken down into a sequence of operations that the computer can perform.
- Then, instructions telling the computer how to perform the operations must be encoded. Sanders (1986, p 540)

Programming languages are vehicles for communication between humans and computers. Coding is when an assembler/compiler accepts *source code* as an input and produce *object code* that is machine dependent.

The problems associated with the coding step of the design step are:

- Style can profoundly affect software quality and maintainability.
- A programming language can limit design to available data structures (data types)
- Technical characteristics of a language can influence the quality of a design.
- Programming language complexity or restrictions can cause problems; source code that is difficult to test or maintain.

Safe Software

Computers are increasingly being used to monitor and control critical functions in such systems as advanced aircraft control, space flights or road traffic control. Most safety-critical activities of complex systems are caused by software controlling mechanical devices. Procedures have to be devised to ensure the safety of human life. The term safety-critical may describe situations where an execution-time failure can result in death, injury, loss of equipment or property, or environmental harm.

The research to provide safe software falls into three main areas:

- software hazard analysis
- verification, validation and assessment
- software design and run-time environments. Littlewood (1987, p 15)

Types of Programming Tasks

Tasks for computer systems will vary in size and type; whether the program you are developing is small or large may be an important factor when choosing a program language.

Most tasks for computers systems fall into one of the following categories:

data logging

data processing

commercial applications

batch processing

scientific and engineering

operating system programming

real-time processing

system control

industrial control

robotics

games programs

The choice of programming language may depend upon the task that the computer has to perform. For example, by tradition, the following languages are preferred (if available):

COBOL	data processing
FORTRAN	scientist and engineers
C	systems programmers
ADA	real-time or embedded computer systems Bell (1987, P85)

Classification of Programming Languages

Computer programming languages may be classified in the following five ways:

FIRST GENERATION LANGUAGES

Machine-level coding (where binary, octal or hexadecimal values are directly inserted into the computer's memory) is still used today to program a computer system. Though, its more likely to be coded using assembly language mnemonics.

SECOND-GENERATION LANGUAGES

Languages that have withstood 30 years of criticism:

COBOL	is still used for business, commercial and data processing applications.
FORTRAN	remains the premier programming language for scientists and engineers.
BASIC	is the most used language on personal computers.

THIRD GENERATION LANGUAGES

Structured programming languages characterised by: strong procedural, data structuring capabilities.

These can be divided into:

general purpose,	C, Pascal and Ada
object-oriented	C++, small talk, Eiffel

FOURTH GENERATION LANGUAGES

Languages with higher levels of abstraction and distinct syntax for control and data structure representation.

Query Languages:

4GL used in conjunction with data bases.

Program Generators:

Third generation-language programs created from using a small set of higher level, more abstract, statements.

Business Information systems applications generate programs in COBOL. Spreadsheets, Database systems, Mackintosh Hypercard allow macros or program statements

Criteria Used to Select a Programming Language

Choosing the most appropriate programming language for a problem is not an easy task. The following list of descriptions may have a greater influence on the choice of language rather that the true criteria for choosing a programming language.

- Organisations have a substantial investment in a particular language. Their programming staff have built up considerable expertise with a particular language.
- Software developers may be contracted to implement a design using a specified programming language.
- Availability of software tools such as language-sensitive editors, debugging systems and project management tools may favour one programming language over another.
- The environment that supports the software may influence your choice of language: For example UNIX has 'C' and MS DOS has BIOS services and DOS functions which provide assembly language programmers with easy access to higher level routines.
- The size of the program may be an important factor.
- A language that is small and simple and can be understood in its entirety enables programmers to become truly proficient and confident, hence, influencing the choice of language.

The true art of choosing a language is to start with the problem, decide what its requirements are and their relative importance. Then match the requirements with the criteria listed below:

Algorithmic and computational complexity

Performance consideration (computer efficiency)

Data structure complexity

Environment in which the software will be executed

Availability of a good computer/assembler or cross compiler/assembler

Debugging tools to protect the user from the details of the hardware

Source code portability

General applications area   Pressman (1992)

Conclusions

After the main functions of each part of the system were listed, data flow diagrams drawn and data types decided upon, then the next major task was to choose the most appropriate programming language for the control programs.

The temperature monitoring device (TMD) and the data transfer device (DTD) control programs have to: input four kilobytes of data, store the data, transfer the data via a serial port, respond to push-button inputs and simply display the status of the device. Note, very little processing of data is done and all the data items and control registers are in byte form. In fact, both control programs will have the following features:

- they must directly control the hardware,
- they will be relatively small in size,
- they will use byte and word data types,
- portability of source code was not required,
- an IBM PC and Motorola evaluation boards provided the programming environment and
- there will be no algorithmic and computational complexity involved.

At the time of creating the JSP diagrams the only available programming languages, to program MC68HC11 devices, were assembly language and `C'. Hence, the author had to choose between them. The author was conversant in programming using both languages on the IBM PC system. After considering the MC68HC11 control program features the author chose a Motorola portable assembler (PASM) as the preferred programming language. The hardware had already been designed and months had been spent on learning the capabilities of the MC68HC11 internal architecture. Hence, being down at the bits and bytes level of design meant that creating the source code using assembly language was the most natural choice at the time. There was no need to shield the programmer from the hardware, there were no complicated data structures used and no complex processing tasks involved. Hence, there was no need to use a high-level language program.

Consequently, the JSP diagrams were created with assembly language programming in mind. Then an efficient programming and debugging environment was created on the IBM PC.

The choice of programming language for the data processing system (DPS) meant that the whole selection process had to performed once again, as follows.

The DPS required a control program to check and store information from the TMS and the DTD. The overall aim of the DPS program was to:

- input a block of information from a serial port,
- the information had to be stored in a file,
- the user had an option of viewing file names in a directory,
- the raw data could be viewed on a VDU or from a hard copy printout.

Hence, a check that the data was safe and ready for data processing could be made by the researcher. Note, that no complicated processing was required from the author, no data types other than: strings for file pointers, 16-bit positive integers for pointers and 8-bit positive integers for data and control register values were required.

Once again, assembly language was chosen for the DPS control program. The following reasons were used to make such a decision:

- The author was very familiar with the MS DOS operating system, the CPU architecture, the system architecture and programming the IBM PC at assembly language level (as well as programming in 'C' and 4GL spreadsheets).
- The author was well aware that MS DOS provides BIOS services that make controlling the hardware a simple task.
- MS DOS provides DOS functions that make file handling easy for the assembly language programmer.
- MASM and TASM assemblers provide a programming and debugging environment comparable to those available to a high-level language programmer.
- User defined labels and macros can make assembly language produce very readable and easy to follow source code. Therefore, debugging is easy as there is a one to one relationship between program labels and JSP labels.

The most pleasing result of this exercise was that there were no problems encountered whilst coding the three control programs. The JSP methodology including assembly language can be recommended to any microcontroller system designer.

It should also be noted that, although there is no portability of source code, the rest of the JSP methodology is portable.

Finally, the coding part of a well designed system is a small mechanical task. In fact, in recent years, research is being done to relieve this task from the designer by making it a software development tool.

4.10. MAINTENANCE AND TESTING INFORMATION

4.10.1. TEST PLAN

Testing involves exercising the program using data similar to the real data that the program is designed to work with in order to observe the program's output and to infer the existence of : errors, inadequacies and anomalies. The plan involves carrying out program testing during implementation and when the implementation is complete.

Although the Temperature Monitoring System was designed using a top-down-approach, the validation of the system uses a bottom-up strategy (as the subsystems of the microcontroller need to be initialised before data transfers can take place). The strategy used for the testing process comes from the book "Software Engineering:" by I. Sommerville. It incorporates five distinct stages in the testing process:

(i) functional testing,
(ii) module testing,
(iii) sub-system testing,
(iv) system testing and
(v) acceptance testing.

(i) FUNCTIONAL TESTING

The software functions are the small units of code that are independent from each other and have their own set of specifications. Each function can be tested as a stand-alone entity. The plan is to define the actions of each device function then describe how it is to be validated.

The testing of the functions for each sub-system of the TMS are described in the following sections of the test plan.

(ii) MODULE TESTING

The modules of each sub-system are also stand-alone units of code. The modules combine the functions in a way that they co-operate with each other to form a task. Each module of a sub-system can be tested on its own. The plan is to define the functions for each module and then describe how the module is to be validated.

The testing of the modules for each sub-system of the TMS are described in the following sections of the test plan.

(iii) SUB-SYSTEM TESTING

The program modules of a sub-system can be put together and tested as a whole unit. Thus, the module interfaces are tested with the assumption that the modules themselves are correct.

In the Temperature Monitoring System the sub-systems include: the TMD, the DTD and the DPS.

(iv) SYSTEM TESTING

This involves the testing of the entire system which comprises of the linking together of the three sub-systems. This testing process is concerned with finding errors in design as well as validating the overall system. It makes sure that the dynamic characteristics of the system match those of the Functional Requirements Specifications.

Testing requires the linking together of:

- the person under test with the temperature monitoring device,
- the temperature monitoring device with the data transfer device,
- the temperature monitoring device with the data processing system,
- and the data transfer device with the data processing system.

(v) ACCEPTANCE TESTING

Acceptance testing is the process of testing the system with real data. Acceptance testing is designed to detect errors in the `Software Requirements Document'. The requirements may not reflect the actual facilities and performance that is required by the user.

DESIGNING TEST CASES

Sommerville (1990), p. 178), states that "planning the testing of each program involves formulating a set of test cases, which are akin to the real data". Test cases should consist of:

- input specifications,
- description of the system functions, and
- a statement of the expected output.

According to Pressman (1987, p. 470) Any engineering product (and most other things) can be tested in one of two ways:

(i) black box testing and

(ii) white box testing.

(i) BLACK BOX TESTING

Black box testing is used when the specified functions that a product is designed to perform are known, and tests can be conducted to demonstrate that each function is fully operational.

Black box testing is conducted at the software interface; test cases demonstrate that software functions are operational, that input is properly accepted, output is correctly produced and the integrity of the system is maintained.

A black box test examines some aspect of the fundamental system model with little regard for the internal logical structure of the software. Black box testing attempts to find:

- incorrect or missing functions,
- interface errors,
- errors in data structure,
- performance errors and
- initialisation or termination errors.

Black box testing was used on the modules of each sub-system, on each sub-system in turn, and the overall system when completed.

(ii) WHITE BOX TESTING

White box testing is used when the internal workings of the product are known, and tests can be conducted to assume that its internal workings perform according to the specifications. White box testing of software is the close examination of procedural details and the testing of the logical paths through the software. It provides test cases to exercise specific sets of conditions and loops of code. The status of the program may be examined at various points to determine if the expected or asserted status corresponds to the actual status of the device. White box testing can:

- guarantee that all independent paths within a module have been exercised at least once,
- exercise all logical decisions (on their true and false side),
- exercise all loops at their boundaries and
- exercise all internal data structures to assure their validity.

4.10.2.1. TESTING THE TEMPERATURE MONITORING DEVICE (TMD)

The TMD uses three areas of storage: a data buffer area, an area to store the fixed parameters of the device, and an area to house the variables used by the control program.

1/ The buffer area which stores all the information gathered during a run of the program includes:

DATEBUF	which stores information relating to the start of the test.
BUFFER	which stores the temperature readings taken during the test.
EVENTB	which records the relative time, with respect to the start of a test, for each event that requires noting during a test.

2/ The fixed parameters, which should be stored in EEPROM, include:

IDENT	the identification of the TMD,
DATA	the start address of the buffer area,
EVENTS	the start address of the events buffer,
DATE	the start address of the date/time buffer.

The other fixed data items are: end of buffer values, start of block marker, end of block marker, timeout values and values for the ADC system.

3/ An area of read/write memory is used to store all the variables that are necessary to run the TMD program. These variables include:

- pointers to buffer areas,
- timer/counter values,
- command flags,
- status flags and
- temporary storage areas.

The Temperature Monitoring Device Program

The TMD program consists of two main control routines, that control the device behaviour, and eight functional routines that control the actions of the device.

The two control routines are:

The RESET module and the PROCESSR module.

The Reset Module

This routine resets the system so that it is in a state of readyness; ready to start logging data from a new test. The major functions of the RESET module are:

(i) to initialise the sub-systems of the microcontroller chip.

(ii) to clear the status and command flags and to clear the data buffer areas of memory.

(iii) to show the status of the device when requested.

(iv) to be ready to ready to accept an upload (start of test) command and respond to it by calling the PROCESSR routine.

The reset module (RESET) makes use of the following functional routines:

INIT1, INIT2 and STATUSR.

The Processing Module

This routine firstly receives the date, time and a start logging command from the data transfer device (or the data processing system). Then secondly cycles round calling a function that detects the need for one of the following four major functions:

(i) to read the next piece of analog data.

(ii) to record the time of an event.

(iii) to show the status of an event.

(iv) to end the test, output the data and to return to the RESET routine.

The processing module (PROCESSR) makes use of the following functional routines:
SYNCR, INPUTR, STATUSR, ANALOGR, EVENTR, OUTPTR. and INIT2

TESTING THE EIGHT FUNCTIONAL ROUTINES

1/ INIT1 Initialise sub-systems routine

input specifications:

The inputs to the sub-systems are initiated by machine code instructions to the I/O registers. The instructions load the following hexadecimal values into the specified registers:

INPUT CAPTURE / OUTPUT STATUS

PACTL	=	80
TCTL2	=	55

ANALOG TO DIGIAL CONVERTER

DDRD	=	00
OPTION	=	A0

SERIAL COMMUNICATIONS INTERFACE

4000	=	FF
BAUD	=	30
SCCR1	=	00
SCCR2	=	03

TIMER COUNTER SUB-SYSTEM

TMSK1	=	00
TFLG1	=	FF

description:

This routine has the task of initialising four of the microcontroller sub-systems. This routine cannot be validated on its own; but it can be validated by the correct operation of the following routines:

expected output:

(i) OUTPUTR may prove that the SCI sub-system has been initialised correctly.

(ii) STATUSR, EVENTR, RESET and PROCESSR routines operating correctly will prove that the input capture and the ouput signals are operating from PORTA correctly.

(iii) PROCESSR and ANALOGR routines operating correctly proves that the TIMER/COUNTER sub-system has been initialised correctly.

(iv) ANALOGR routine converting the analog signals to digital values proves that the ADC sub-system has been initialised correctly.

2/ INIT2 Initialise variables routine

input specifications:

data buffer address	=	DATA
events buffer address	=	EVENTS
date/time buffer address	=	DATE
variables start address	=	CLEAR

description:

This routine clears all the buffers, command flags and status flags. It also sets the buffer pointers with their initial values.

expected outputs:

DATA	to	BUFFMAX	=	00
EVENTS	to	PTREMAX	=	00
DATE	to	DATEMAX	=	00
CLEAR	to	VARYMAX	=	00

3/ STATUSR Display status routine

input specifications:

IC2	=	`1'	pressing the STATUS push button makes IC2 i/p		
	=	+5V			
MONITOR	=	FF	or	00	monitoring data flag
MEMFULL	=	FF	or	00	memory full flag
BATTERY	=	FF	or	00	battery low voltage flag
TRANSFER	=	FF	or	00	transferring data flag

description:

This routine is normally initiated by pressing the STATUS push button. The logic of the code tests four status flag bits in turn. If a flag bit is set the appropriate LED will be illuminated. If the flag bit is zero the appropriate flag bit is switched off. The output to the 4 LEDs is sent to PORTA for a period of 3 seconds, then cleared to save power.

The STATUSR routine is called from both of the two main modules.

expected output:

This routine outputs a logic level of `1' to illuminate the following LEDs:

MONITOR
MEMFULL
BATTERY
TRANSFER

Note, if all the status flags are set to $FF at the same time (which cannot occur in the normal correct execution of the program), this situation indicates that there was an error when transferring data.

Note, if all the status flags are reset to zero (all the LEDs are switched off) this indicates that the device is in the ready mode.

4/ SYNCR Upload date, time and start command routine

input specifications:

IC3	=	`1'	pressing the UPLOAD push button makes IC3 i/p
	=	+5V	

SCI = `**ddmmyyhhmmss#' a block of ASCII characters

**	start of block marker
dd	2 digit ASCII hexadecimal value for the day
mm	month
yy	year
hh	hour
mm	minute
ss	second
#	end of block marker (start monitoring command)

description:

This routine organises the correct transfer of a block of ASCII characters from, either the DTD or the DPS, to the DMD. The data transfer is initiated by the pressing of the UPLOAD push button on the DMD and a down land command from either the DTD or the DPS.
This routine checks the start and end of block markers (* and #) and checks that the number of data bits does not exceed 12. There is also a one minute timeout between pressing the UPLOAD push button and detecting the start of block marker, and a timeout between each character. If an erroneous transfer occurs then all the device status flag bits are set to FF.
When a successful transfer is completed the date and time values are stored in the DATEBUF memory area the input capture flag register and the data transfer status flag are cleared.

expected output:

DATEBUF	=	dd mm yy hh mm ss (date and time information)
TRANSFER	=	00

or

DATEBUF	=	00 00 00 00 00 00 (no date and time information)
MONITOR	=	FF
MEMFULL	=	FF
BATTERY	=	FF
TRANSFER	=	FF

5/ INPUTR Checks for push button commands and 10 min. timeout

input specifications:

TENMIN	=	TILIMIT	or a value between 0 and TILIMIT	
IC1	=	'1'	or	'0'
IC2	=	'1'	or	'0'
IC3	=	'1'	or	'0'

description:

This routine is called from the PROCESSR routine thousands of times per second. It is used to detect immediately one of the following occurrences:

(i)	a 10 minute timeout;	if not increment TENMIN value.
(ii)	an event needing to be recorded;	if yes set EVENT flag = FF else = 00.
(iii)	a display status request;	if yes set DISPSTAT flag = FF else = 00.
(iv)	a down load request;	if yes set DNLOAD flag = FF else = 00.

expected output:

TENMIN	=	TILIMIT or a value between 0001 and TILIMIT		
EVENT	=	FF	or	00
DISPSTAT	=	FF	or	00
DNLOAD	=	FF	or	00

6/ ANALOG Routine to read the next piece of data from the ADC sub-system

input specifications:

READFLAG	=	FF	read ADC command flag
ADR1	=	digitised temperature reading	
ADR2	=	digitised battery voltage	

description:

The main aim of this routine is to read the current temperature value and the current state of the battery voltage from the ADC. To do this the ADC is programmed for a multi-channel single scan mode of operation. Then channels 0 and 1 are read and the following tasks are performed:

(i) The temperature reading is stored in the data buffer. Then a test for the buffer being full is made. When the data buffer is full the MEMFULL status flag is set to FF.

(ii) The battery condition reading is checked against a minimum value (BATLOW). When the battery condition is found to be low the BATTERY flag is set to FF.

(iii) The number of temperature readings value (AMOUNT) and the buffer pointer (BUFFPTR) are incremented if the MEMFULL flag is not set.

expected output:

BUFFER	has a new temperature reading added to it.
TEMP2	has the current battery voltage value.
TEMP3	a coded value for the current temperature reading.
BUFFPTR	is incremented
AMOUNT	is incremented
MEMFULL	is set to FF if BUFFPRT = BUFFMAX value, else = 00.
BATTERY	is set to FF if battery voltage is below the BATLOW value, else 00.
READFLAG	is cleared.

7/ EVENTR Routine to record the time of an event

input specifications:

EVENT	=	FF	event command flag

description:

This routine is called from the PROCESSR routine to record the elapsed time from the start of a test (the AMOUNT value) when an event command is issued. The 16-bit value is stored in the events buffer (EVENTB). A test for the events buffer being full is also made. If the buffer is full the EVENTFUL flag bit is set to FF.
The event buffer pointer is incremented by 2 and the event command flag is reset to zero.

expected output:

EVENTB	has a new 16-bit value added to it
EVENTFUL	= FF if EVENTPTR = PTREMAX, else = 00
EVENTPTR	is incremented by 2
EVENT	is cleared

8/ OUTPUTR Routine to down load results to the serial port

input specifications:

DNLOAD	=	FF	end of test; down load data to serial port

description:

Firstly, the transmitter part of the SCI sub-system is enabled. Then the following information is sent to the serial port (Data Transfer Device or the Data Processing System):

- a start of block marker '*',
- the device identification,
- the date and time that was originally uploaded,
- the block of temperature readings,
- the block of event times,
- the end of block marker '#'.

Then the input capture flag register and the TRANSFER flag are cleared.

expected output:

The following information is sent to the serial port at the 9600 baud rate:

```
*
IDENT
dd mm yy hh mm ss
a block of temperature readings
a block of  event times
#
```

TFLG1	the Timer input capture flag register is cleared
TRANSFER	the transfer status flag is cleared

9/ RESET Initialisation Module

input specifications:

IC2	=	`1'	to detect a request to show the status of the device
IC3	=	`1'	to detect the request to start logging data

description:

This module is used to reset the temperature monitoring device system so that it is in a state of readiness. So that it is ready to log data for a new test. The RESET module calls up functional routines to perform the following tasks:

(i) to initialise the sub-systems of the microcontroller.

(ii) to clear the three main buffer areas of memory,
to clear the status and command flag bits ready for use, and
to initialise all the program variables.

(iii) to allow the status of the device to be shown at any time.

(iv) to be ready to accept an UPLOAD command (a start of test command) from the UPLOAD push button, and respond by calling the PROCESSR module.

expected output:

calling the STATUSR routine in response to IC2 = `1' (+5V)
all the status flag bits should be reset to zero
the input capture flag register should be cleared

calling the PROCESSR routine in response to IC3 = `1' (+5V)
the MONITOR flag bit should be set to FF
the input capture flag register should be cleared

10/ PROCESSR The Data Logging Module

input specifications:

INPUTR routine detects the following inputs:

(i) TOF = `1' to detect a timer overflow and increment TENMIN value.
(ii) TENMIN=TILIMIT to detect a read a new temperature value.

(iii)	IC1	=	`1'	to detect a request to record the time of an event,
(iv)	IC2	=	`1'	to detect a request to show the status of the device,
(v)	IC3	=	`1'	to detect the request to end logging, and output results.

description:

This module receives the date, time and start logging command, from the serial port, via the SYNCR routine, then cycles round calling 5 routines which enable the following 4 main tasks to be performed:

- to read the next temperature value,
- to record the time of an event,
- to show the status of the device, and
- to end the test and output the results to the serial port.

expected output:

calling the ANALOG routine in response to READFLAG = FF

calling the STATUSR routine in response to DISPSTAT = FF

calling the EVENTR routine in response to EVENT = FF

calling the OUTPUTR routine in response to DNLOAD = FF
call INIT2 routine
return back to the RESET routine

4.10.2.2. TESTING THE DATA TRANSFER DEVICE (DTD)

The DTD uses three areas of storage: a data buffer area, an area to store the fixed parameters of the device, and an area to house the variables used by the control program.

1/ The buffer area which stores all the information gathered during a run of the program includes:

DATEBUF	which stores information relating to the start of the test.
BUFFER	which stores the temperature readings taken during the test.
EVENTB	which records the relative time, with respect to the start of a test, for each event that requires noting during a test.
IDENT	the identification of the TMD,

2/ The fixed parameters, which should be stored in EEPROM, include:

DATA	the start address of the buffer area,
EVENTS	the start address of the events buffer,
DATE	the start address of the date/time buffer.
BUFFMAX	the end of the data BUFFER value,
PTREMAX	The end of the EVENTB value,
DATEMAX	the end of the date and time buffer (DATE).
BEGCODE	the other fixed data items are: end of buffer values, start of block marker,
ENDCODE	end of block marker, timeout values and values for the ADC system.
TILIMIT	the timeout value, used to check the maximum no. of TOFs before a bad serial transfer is declared.

3/ An area of read/write memory is used to store all the variables that are necessary to run the TMD program. These variables include:

- messages for the LCD screen,
- pointers to buffer areas,
- timer/counter values,
- command flags,
- status flags and
- temporary storage areas.

The Data Transfer Device (DTD) Program.

The DTD program consists of 5 main control routines, that control the functions of the system that the user can select. The DTD program, at a lower level, consists of 9 major functions that can be selected by the user, 2 minor functions that input information from the user, and 19 basic functions that control the actions of this device.

The 5 main control routines (modules) are:

MAIN, COMMANDR, MENU1R, MENU2R and MENU3R.

The 9 major functions can be sorted into 3 categories:

To display information:

TIMER, STATUSR and DATER.

To transfer information:

STARTR, UPLOADR and DNLOADR.

To change information:

RESETR, DATER and CLEARR.

The 2 minor functions are:

KBDTIME and KBDDATE.

The basic functions can be sorted into 3 categories:

The system initialisation routines:

INIT1R, INIT2R, INIT3R and INIT4R.

The keyboard control and data conversion routines:

INPUTR and CONVERT.

The 19 basic functions are:

OUTPUT, OUTPUT2, SCREEN, DELAY,
SHOW, BLANK, DISPLAYR, TLC, CURSOR,
WRITE, WRITEHEX, DUMP, AND VIEWLINE.

The MAIN module:

This routine initialises the LCD module and the microcontroller sub systems that are used by the DTD. It also clears all the data buffers and variables that are used by the DTD. The major functions of the MAIN routine are :

(i) to initialise the SCI sub system,

(ii) to initialise the LCD module,

(iii) to initialise the input capture sub system, and

(iv) to clear all the data buffers and variables used by the system.

The MAIN module makes use of the following functional routines:

INIT1R, INIT2R, INIT3R and INIT4R.

The COMMANDR module:

The COMMANDR module is a high level control routine that displays, and enables the user to select, the 3 main functional areas of this device; the display, the transfer of the changing of information within the device.

This module inputs information from a hexadecimal keypad which enables it to give control over to one of the lower level control modules: MENU1R, MENU2R or MENU3R.

This module makes use of the following modules:

MENU1R, MENU2R and MENU3R.

This module also makes use of the INPUTR functional routine to input from the KBD.

The MENU1R module:

This module shows a screen with 3 sorts of information that the user can choose to display on the LCD. The user can either view:

- the status of the device and uploaded information,
- the `start of test' date and time prior to a start of a new test, or
- the data that has been uploaded from a TMD.

This module inputs the user's choice, then gives control over to the appropriate routine, making use of the following functional routines:
TLC, DISPLAYR, INPUTR, STATUSR, TIMERR and DATERR.

The MENU2R module:

This module shows a screen on the LCD that displays the 3 choices of serial data transfers that the DTD has been designed to perform:

- to enable the TMD to start a new test,
- to upload information from the TMD, or
- to down load data, gathered from a test, to the data processing system (DPS).

This routine also enables the user to input, via the hexadecimal keypad, their choice of transfer. This module then calls up the appropriate major function to perform the task.

This module makes use of the following functional routines:

TLC, DISPLAYR, INPUTR, STATUSR, TIMERR and DATERR.

The MENU3R module:

This module shows a screen with 3 sorts of information that the user can choose to change:

- the 'start of test' date and time,
- the clearing of all the data buffers and variables, or
- the complete reseting of the DTD system.

This routine also enables the user to input, via the hexadecimal keypad, their choice of change. This module then calls up the appropriate major function to perform the task.

This module makes use of the following functional routines:

TLC, DISPLAYR, INPUTR, STATUSR, TIMERR and DATERR.

4.10.2.3. TESTING THE DATA PROCESSING SYSTEM (DPS).

The DPS uses three areas of storage: a data buffer area, an area to store the messages for the VDU screen, and an area to house the variables used by the control program.

1/ The buffer area which stores all the information gathered during a run of the program includes:

IDENTV	the identification of the TMD that gatered the data.
DATEBUF	which stores information relating to the start of the test.
BUFFER	which stores the temperature readings taken during the test.
EVENTB	which records the relative time, with respect to the start of a test, for each event that requires noting during a test.

2/ The names of the messages used by the dps program are:

MSG1	serial port error message
MSG2	printer port error message
MSG3	return to DOS message
MSG4	file transfer error message
MSGMENU	main menu screen
MSGINPUT	uploading serial data screen
MSGRAW	displaying raw data screen
MSGDIR	displaying a directory of filenames
MSGLOAD	loading a data file screen
MSGSAVE	saving a data file screen
MSGPLOT	plotting the results screen
MSGEXIT	exit to DOS screen

3/ An area of read/write memory is used to store all the variables that are necessary to run the TMD program. These variables include:

- pointers to buffer areas,
- timer/counter values,
- command flags,
- status flags and
- temporary storage areas.

The Data Processing System Program

The DPS program consists of two main control routines. One that initiates the input/output interfaces and the data area of memory, the other allows the user to choose one of 6 major functions that they can ask the system to perform.

The two control routines are:

the MAIN module, and the COMMANDR module.

The MAIN module description.

This module is responsible for resetting the system interfaces and initialising the data area of memory. The program is then in a state of readyness, so that users can select the tasks which they want the system has to perform.

The major functions of the MAIN routine are:

(i) to initialise the serial port.

(ii) to initialise the printer port.

(iii) to initialise the screen and display the main menu.

(iv) to initialise the counters, pointers, variables and buffers that will be used by the DPS routines.

(v) to hand control over to the COMMANDR module.

The MAIN module makes use of the following functional routines:

INIT1R, INIT2R, INIT3R and INIT4R, then gives control to COMMANDR.

The COMMANDR module description.

This module waits for a KBD input in response to the main menu choices, which are displayed on the VDU screen. The major functions that can be selected by the user perform the following tasks:

(i) Upload information from the TMD or the DTD via the serial port.

(ii) Display the filenames from a specified secondary storage directory.

(iii) View parts of the information that has been loaded into the primary memory.

(iv) Load the raw data (from a TMD test) from a specified secondary storage file.

(v) Save the TMD test information from memory to a specified secondary storage file.

(vi) Plot the results from a TMD test, onto a printout or onto the VDU screen.

(vii) Exit from the program.

The processing module COMMANDR makes use of the following functional routines:

INPUT, DIR, RAW, SAVE, PLOT, OLD, and EXIT.

TESTING THE 11 MAJOR FUNCTIONAL ROUTINES

1/ INIT1R Initialising the serial port.

With assembly language programming on the IBM PC, the inputs, in many cases, are the parameters that are passed to the Basic Input/Output Subprograms (BIOS) service routines, and to the Disk Operating System (DOS) functions. These parameters are passed through the CPU 16-registers: AX, BX, CX, DX, SI AND DI, and the CPU 8-bit registers: AH, AL, BH, BL, CH, CL, DH and DL.

input specifications:

AH	=	00	service number, to initialise the serial port
AL	=	E3h	9600 baud, 8-bits, 1 stop bit, no parity
DX	=	0000	serial port number also
AH	=	01	send one character service
AH	=	02	receive one character service
AH	=	03	get serial port status service

description:

This routine has the task of initialising the serial port (COM1) of the system by the use of a BIOS service routine. The status of the serial port is then checked using a different BIOS service.

expected output:

The status of the serial port is passed to the program via the CPU register AH.

AH	bit 0	=	(i/p) data ready
	1	=	overrun error
	2	=	parity error
	3	=	framming error
	4	=	break detected
	5	=	(o/p) transfer register empty
	6	=	shift register empty
	7	=	timeout error

2/ INIT2R Initialise the printer port.

input specifications:

	AH	=	01	initialise the printer
	DX	=	0000	LPT1
also				
	AH	=	00	get printer status
	AH	=	02	send one character

description:

This routine has the task of initialising the printer (LPT1) of the system. The printer status can then be checked for error conditions and to check whether it is ready or not.

expected output:

The response to checking the printer status is via register AH:

AH	bit	3	I/O error
		4	selected
		5	out of paper
		6	acknowledge
		7	ready

3/ INIT3R Initialise the program variables.

input specifications:

The start address of the following data buffers are put into the CPU index register SI: one at a time, in turn, and used to clear the appropriate data buffer.

IDENTV
DATEBUF
BUFFER
EVENTB

The size of the data buffer is put into the CPU count register CX.
The value 00h is put into the 8-bit accumulator AL

description:

Four similar routines are used to store the value zero into each buffer area. This is done to clear out any previous information that may be stored there.

expected output:

IDENTV
DATEBUF
BUFFER
EVENTB

4/ INIT4R Initialise the VDU screen.

input specifications:

The following parameters are passed to the BIOS service routine (INT 10h).

AH	=	00	service to set the screen mode
AL	=	03	text mode 3, 80 character, 25 lines

The address of the message to be displayed at the end of this initialisation function is loaded into the SI register and passed to the DISPLAY routine.

SI = address of MSGMENU

description:

There are two tasks for this routine. The first is to ask BIOS to clear the screen, the second is to call the DISPLAY routine to display a menu on the VDU screen. The menu shows the options open to the user, and describes what the user has to do to choose one of the options.

expected output:

The screen is cleared of previous information, then the message MSGMENU appears on the VDU screen

5/ INPUT Upload TMD information from the serial port.

input specifications:

SI	=	address of MSGINPUT

A stream of serial data is input from the serial port (COM1) using a BIOS service routine. The BIOS service requires the following parameters to be passed to it for the reciept of each character:

AH	=	02	service number, for the reciept of one character
DX	=	0000	serial port nuber for COM1

Also, to prevent the system locking up (in the case of bad data, or no data) the user can input any character via the keyboard. Hence, BIOS also requires (for its INT 16H service routine):

AH	=	01	service to detect a KBD keystroke.

ROW and COL variables are also used to input the positions on the screen for the resultant messages.

description:

This routine is designed to input, from either the TMD or the DTD, a block of serial data that has been sent in a particular format. The data is not validated on receipt but certain control characters are checked:

*	=	the start of block marker
@	=	the end of data marker
#	=	the end of block marker

The user can input a KBD keystroke to end the search for serial input, if they believe that something has gone wrong.

A successful transfer results in messages on the screen, showing the identification of the TMD that gathered the information and the starting date of the test.

expected output:

IDENTV	=	identification of the TMD
DATEBUF	=	the starting date and time of the test
BUFFER	=	the items of data
EVENTB	=	the recorded event times

6/ DIR Display a directory of files on the VDU screen

input specifications:

SI	=	address of MSGDIR

The BIOS routines (INT 10h) uses the following parameters that are passed to them:

AH	=	02h	service to move the cursor
AH	=	0Eh	service to write one character to the screen
DH	=	ROW	position on the VDU screen
DL	=	COL	" "
BX	=	0000	display page

The DOS functions (INT 21h) require the following parameters to be passed to them:

AH	=	1Ah	function to establish a data transfer area (DTA)
	=	4Eh	function to find the first matching filename
	=	4Fh	function to find the next matching filename
DX	=	DTA	a data transfer area (DTA)
	=	PATH	the specified drive, path and filename

description:

This routine asks the user to input, via the keyboard: a drive, path and filename. Then 3 DOS functions are used to find all the matching filenames in the specified directory. Each file in turn is placed into the DTA. A routine copies each filename, in turn, and displays it on the VDU screen for viewing purposes.

expected output:

The VDU screen will display either:

(i) an error message

or

(ii) a table of filenames on the VDU screen.

Note, no information is stored in the computers data area of memory except:

PATH	=	drive, directory path and a filename

Note, DOS allows the ? and * wild-cards.

7/ RAW Routine to display the raw data from a TMD test

input specifications:

SI	=	address of MSGRAW
ROW	=	position on the screen
COL	=	" "
IDENTV	=	identification of the TMD
DATEBUF	=	the starting date and time of the test
BUFFER	=	the items of data
EVENTB	=	the recorded event times

description:

This routine allows the user to validate the information received from a TMD test by viewing the data on the VDU screen. The TMD identification, the starting date of the test and the first240 bytes of the data from the test are displayed on the screen. Then a message describing how to make the program return to the main control routine is displayed on the screen.

expected output:

IDENTV	=	identification of the TMD,
DATEBUF	=	the starting date and time of the test,
BUFFER	=	the first 240 items of data, in hexadecimal format, 15 rows of 16 data items.

8/ SAVE Routine to save the TMD information.

input specifications:

SI	=	address of MSGSAVE
DX	=	address of the DTA
DI	=	address of the PATH
AH	=	1Ah DOS function for establishing a DTA
	=	3Ch DOS function for creating a new file
	=	40H DOS function for writing characters to a file
IDENTV	=	identification of the TMD
DATEBUF	=	the starting date and time of the test
BUFFER	=	the items of data
EVENTB	=	the recorded event times

description:

This routine allows the user to save the information received from a TMD test into a file on a secondary storage device. It does this by clearing the VDU screen and displaying MSGSAVE. Then asks the user to input a complete file specification, including the name of the file where the test information is to be stored. The TMD test information is then transferred from the primary memory, into the specified file in a secondary storage.

In case of a file transfer error, a message will be displayed on the VDU screen.

expected output:

The VDU screen is cleared, then the message MSGSAVE appears on it.

A new file will be created if the file specification is new.

The TMD test information is then transferred from the primary memory, into the specified file.

In case of a file transfer error, a message will be displayed on the VDU screen.

9/ OLD Routine to load TMD test information from a specified file.

input specifications:

SI	=	address of MSGOLD	
AL	=	access rights, read only	
BX	=	handle of opened file	
CX	=	number of bytes to read from a file	
DX	=	address of PATH	
AH	=	3Dh	DOS function to OPEN a file
	=	3Eh	DOS function to CLOSE a file
	=	3Fh	DOS function to READ from a file
ROW		position on the VDU screen	
COL		" " "	
DX	=	address of	IDENDTV
		"	DATEBUF
		"	BUFFER
		"	EVENTB

description:

This routine allows the user to load old TMD test information, from a file in a secondary storage device, to the appropriate buffers that are accessable to the DPS program. The program does this by creating a new screen with instructions for the user. The user is asked to input a file specification, via the KBD into the variable PATH.
This routine calls other routines:

- to get the TMD identification,
- to get the starting date of the text,
- to get the many items of data, and
- to get the times of the recorded events.

The TMD identification and starting time are displayed on the VDU screen to inform the user of a successful load. MSG3 is then displayed on the VDU screen to inform the user how they can return to the main control routine COMMANDR.

If an unsuccessful loading has been detected, then an error message is displayed.

expected output:

The VDU screen is cleared then MSGLOAD is displayed

Then either:
The data area buffers: IDENTV, DATEBUF, BUFFER and EVENTB are loaded with information from the specified file.

The contents of IDENTV and DATEBUF are displayed on the VDU screen.

or

In the case of a file transfer error, MSG4 is displayed on the VDU screen.

also

A message informing the user how they can return to the main control routine COMMANDR will be displayed on the VDU screen.

10/ PLOT Routine to create a hardcopy/softcopy of the results from a test.

input specifications:

SI	=	address of	MSGPLOT
	=	"	MSGVDU
	=	"	MSGPRN
IDENTV	=	identification of the TMD	
DATEBUF	=	the starting date and time of the test	
BUFFER	=	the items of data	
EVENTB	=	the recorded event times	

description:

This routine allows the user the option of selecting either: a softcopy or a hardcopy of the results from a TMD test. This routine firstly creates a new screen MSGPLOT which informs the user of their options. The routine waits for a keyboard response. Then the raw data from the memory buffers (BUFFER and EVENTB) are displayed graphically on the VDU screen or onto a paper printout.

expected output:

This routine either:

(i) A graphical plot (one screenfull at a time) of the data from a TMD test.

or

(ii) A complete hardcopy printout of a plot (temperature versus time).

also

A message informing the user how they can return to the main control routine COMMANDR will be displayed on the VDU screen.

11/ EXIT Routine enabling a return to DOS.

input specifications:

SI	=	address of MSGEXIT
AH	=	00 BIOS service number, for KBD software interrupt 16h

description:

This routine displays the message: "Do you really want to EXIT the program".
"Type in `y' or `n' "

In response to the question the program control is either: returned to the program, or returned to the operating system (DOS).

expected output:

Either:

(i) a return to DOS,

or

(ii) a return to the COMMANDR module.

4.11 PROGRAM TESTING

Each part of the temperature monitoring system was tested separately, then the complete system was tested as a whole. The order in which the sub-system's programs were tested is as follows:

1. the temperature monitoring device (TMD),
2. the data transfer device (DTD) and
3. the data processing system (DPS).

4.11.1 TESTING THE TMD SUB-SYSTEM

A circuit board had to be designed and constructed before the TMD program could be tested. This circuit board contained the following:

- a +5V and 0V power supply,
- two 10K ohm potentiometers which represented the two analog signals to be monitored (body temperature and battery voltage),
- three push button switches labelled: ***up/dn***, ***st*** and ***ev*** (upload/download, status and event),
- four LED circuits designed to show the status of the device (see figure 12) and
- a 60-way cable and connector enabling the circuit board to be attached to the TMD evaluation board.

The TMD program was tested from top to bottom. The testing processes began by examining the data flow diagrams, Jackson Structure Diagrams and the maintenance and test information.

Firstly, the initialisation routines were tested to ensure that: all the data buffers were available for use, all the program variables were initialised and all the appropriate microcontroller subsystems were ready to be used.

Secondly, the higher-level control mechanisms of the TMD program were tested. This enables control of the program to be transferred from the READY module to either the SHOW or START module. Hence, the system was ready to either display the status of the TMD or start logging data. This control was activated when either the status (st) or the upload (up/dn) pushbutton was pressed.

The mid-level control of the TMD was tested next. This also required the testing of bush buttons and the display of status information. But most importantly, a thorough testing of the ten-minute timeout, analog input and data storage processes was completed.

Finally, the downloading of a set of results was tested. This was achieved by reading the data from the IDENTS, DATA and EVENTS buffers and ensuring the information was output to the serial port of the microcontroller. An IBM PC running a serial communications program was used to receive and display the information.

The technique for testing the correct execution of the program modules and routines was to press pushbuttons, ensure responces to timeouts and to stop program execution at the inserted breakpoints. Then the appropriate flag bits, status values and items of data were checked.

The complete TMD program controlling the hardware was then tested thoroughly. The analog values going into the system were recorded on paper and checked against the results sent out to the IBM PC system. The testing process include implementing each of the predefined test cases and then checking for the expected outputs.

4.11.2 TESTING THE DTD SUB-SYSTEM

Before testing of the DTD program could take place the following devices had to be connected to the evaluation board:

- a four line by 20 character LCD module,
- a hexadecimal keypad circuit and
- an IBM PC with a serial communications program. (see figure 13)

Firstly, the initialisation programs had to be tested to ensure that the microcontroller subsystems and the LCD module were programmed correctly and that the buffers and variables were initialised. Note, the MAIN module of the DTD program makes sure that the microcontroller is ready for action.

The second stage of the testing checked that the high-level transfer of control mechanism linked the COMMANDR module to the MENU1, MENU2 and MENU3 modules. This involved the pressing of keys on the hexadecimal keypad and tracing the paths to the appropriate program module. Execution started from the entry point of COMMANDR and was ended by one of the breakpoints which were inserted at each entry point of the lower-level subroutines.

The third and most time consuming task was to write all the lowest-level subroutines that performed most of the computational and output work. These routines are called by all the mid-level modules that control the input and output of information.

The next stage was to test the nine mid-level modules individually. This involved making sure that all the correct messages were displayed on the LCD module's screen and that the correct information was input, stored, displayed and output.

Finally, the complete DTD sub-system was validated to ensure that al the functions of the device worked as they were designed to do. This involved:

- time values being input and stored,
- the start of test information being downloaded to an awaiting IBM PC system,
- a block of data being received (uploaded) from the IBM PC,
- data being viewed (a screen-full at a time) and
- a block of data being retransmitted (downnloaded) to another IBM PC system.

When all the functions of the DTD had been tested thoroughly the sub-system was deemed to be validated. Consequently, blocks of data, of a particular format, could be input, viewed and output as the design of the DTD specified.

4.11.3 TESTING THE DPS SUB-SYSTEM

The data processing system (DPS) required a TMD, DTD and an IBM PC to be connected by their serial ports before testing could begin. (see figure 11) Note, a second IBM PC could just as easily act as a TMD or DTD for the purpose of testing the DPS. Note, the following tests took place:

The first task was to check all four DPS initialisation routines called by the MAIN routine. Thus, the data buffers were cleared, variables initialised, the screen mode set, a message displayed and the serial port parameters programmed into the system.

The second stage of testing was to check the transfer of control mechanism in the COMMANDR module. This involved the pressing of keys on the keyboard, decisions being made and execution stopping where breakpoints had been placed at the entry points of the mid-level modules.

Note, as MS DOS provides all input/output subroutine (BIOS) services and disk operating system (DOS) functions, no general purpose low-level routines needed to be written.

Then each mid-level module was tested separately in order to:

- input a block of data,
- view the raw data,
- look at a directory of files,
- save the raw data on a file,
- retrieve raw data (old results) from a file and
- plot the results graphically, either on a VDU screen or on a hardcopy printout.

Finally, the complete DPS program was tested thoroughly and was found to work as expected.

4.11.4 TESTING THE TEMPERATURE MONITORING SYSTEM

The TMD, DTD and the DPS were all connected together, by their serial ports, in order to input data and to check that the correct transfers of data took place. The following checks were made to confirm that the system would perform as expected.

The time and date of starting the testing of the TMS was entered into the DTD. Then the TMD program was given a start signal from the DTD.

Temperature values and event times were recorded by the TMD and manually on paper. During the recording session the status of the TMD was displayed when requested.

After a known amount of time, the TMD recording session was ended and the stored information was transferred from the TMD to the DTD.

The data transfer device was made to display the raw data and event times to check the performance of the trial. As expected the results on paper matched the results shown on the DTD's liquid crystal display screen.

The next step was to transfer the stored information from the DTD to the DPS.

Finally, the raw data from the recording session was displayed on the DPS screen. The data values and event times matched those that were recorded on paper. Consequently, the testing session was deemed to be successful.

CHAPTER FIVE

5 CONCLUSIONS

5.1 Discussion

the problem
the need
the feasibility
the cost effectiveness

5.2. System Design

system description
temperature monitoring device
data transfer device
data processing system
analysis techniques
findings from library search
a JSP methodology
the software development environment

5.3. System Testing

inherent testability and maintainability
types of testing
sequence of testing

5.4 Suggestions for Further Considerations

introduction
fabricating miniature devices
power supply and battery use
similar devices
convenient body sites
need for networking devices
need for 32-bit devices

5 CONCLUSIONS

5.1. DISCUSSION

There was, and still is, a need for monitoring the effects of circadian rhythms in humans. Lack of sleep, or sleep at the wrong time, affects work performance or even causes absenteeism. Body temperature is known to be a method of determining when the performance of our body is at a peak and when it is informing us that it is time to sleep. A small portable temperature data logging device would be invaluable in this field of research. This thesis is concerned with designing, implementing and testing such a device. At the commencement of this Master's work, investigations indicated that there was no suitable device commercially available.

The author found that a circuit can be designed around an integrated circuit temperature sensor (LM 35) and a precision thermistor (YSI 44002) to provide temperature readings within a resolution of 0.1 degrees Celsius. Either of these circuits could provide a suitable analog input for a microcontroller-based device. The axillary site was considered to be the best non-intrusive body site for measuring body temperature.

Investigations demonstrated that microcontrollers provide a low-cost single-chip solution to this problem. All the complexity of the hardware and hardware interfaces are encapsulated within a single chip. Which means that a temperature data logger can be made small enough to be worn without interfering with normal body movements.

Further investigations indicated that an 8-bit microcontroller would provide the necessary resources for the data logger. So, an in-depth study of 8-bit microcontrollers was made. Comparison of several 8-bit microcontroller devices resulted in a report and the enclosed tables of specifications (table 1.1 and 1.2). Analysis of the list of specifications (from six devices) shows the similarities and differences that exist within 8-bit microcontrollers. As a consequence, the Motorola MC68HC11 series of microcontroller was found to be the most suitable device for controlling the input of analog signals, storage of readings over a four week period and serial output of the temperature values. It should be noted that Motorola supports its microcontrollers with an environment, in the form of evaluation boards and software, which is necessary to develop the device's control programs.

5.2. SYSTEM DESIGN

The temperature monitoring system (TMS) is a system that is designed to monitor a person's body temperature, enable the recordings to be stored in the monitoring device's memory, allow the recorded data to be transferred to a data processing system and then allow the data to be processed and filed away permanently. The TMS comprises of three, independently controlled, main parts; namely:

- a temperature monitoring device (TMD) worn by the person under test,
- a data transfer device (DTD) and
- a data processing system (DPS).

The TMD which needs a start signal and time of test information (from a DTD or a DPS) was designed to record temperature readings, every ten minutes, for a period of up to 4 weeks. This device is small enough to be worn throughout the period of time the person is taking part in a test. That is, up to a period of four weeks.

If test conditions required a remote site then the small and inexpensive DTD could be used to start the test and later upload the recorded information from the TMD. The DTD itself, or a memory module containing the information, would be posted to the researcher who would enter the data into a DPS.
The DPS enables the researcher to, first of all, check the raw data and then store it in a permanent file. The researcher can analyse the results later with whatever software he/she requires.

An important part of this work was to produce a system that was designed to software engineering standards. Hence, a search for a suitable design methodology was conducted.

Investigations revealed that there were three main streams of analysis techniques in software engineering:

- structured analysis,
- object-oriented analysis and
- formal specification techniques.

Structured analysis, and in particular, a JSP methodology was found to be the most suitable design methodology for microcontroller-based systems. It provided a highly systematic approach to software design as, it requires a problem to be well defined, it can be used for real-time systems, it is language independent and it creates a very graphical hierarchical solution to a problem.

Designers need to be made aware that system usefulness only lasts while it satisfies requirements and that user requirements rarely remain static. Hence, a system design must have software that is portable, reusable, have maintainability and have extendability. The design should allow for hardware or software to be taken apart, modified and then reassembled. A JSP methodology can be used to produce software with such properties and it also tries to ensure that untouched modules stay in tact.

An extensive library search was made in order to find out which methodologies have been used for microcontroller designs. The findings from the search informed us that:

- there was a lot of material on software design methodologies,
- there was a growing interest in microcontroller-based designs, but
- the majority of articles describing microcontroller designs gave very little emphasis to a design methodology.

In fact no articles, or dissertations on CD-ROM, had any information relating to "microcontrollers and Jackson structured programming (JSP) methodologies". The author therefore adapted the JSP methodology to small system designs incorporating microcontrollers and proposes this method as a suitable development platform for microcontroller-based system design.

The JSP design methodology required detailed hardware specifications to be made. Hence, the TMS which was defined to consist of three main components; namely, the TMD, the DTD and the DPS, had to be analysed further. Each main part of the TMS had its functions listed so that the required hardware components could be assessed. The Motorola MC68HC11 microcontroller, that was selected because its internal hardware could perform the necessary expected functions of each device, required a thorough knowledge of its sub-systems down to the register level. The registers being used by the control program software.
An IBM PC and peripherals was the only hardware necessary for the DPS.

A top-down approach was taken to produce the control programs. Firstly, the control programs were described as a hierarchy of data flow diagrams (DFDs). Then, in turn, each DFD was translated into a Jackson structure diagram (JSD). The JSDs included the conditions that were required to move control from one module of the program to another module and a list of functions that were needed to be executed in order to perform tasks. Finally, the JSDs were transformed into assembly language instructions.

The actual coding process included the same names, for subroutines and labels, as those used by the DFDs and JSDs. This made sure that the software would be easy to test and maintain.

An efficient software development environment was created so that programs could be developed and debugged via an IBM PC system. The environment consisted of an editor, assembler, linker, ubuilds and communication programs. The aforementioned development software tools were called from a MS DOS batch file program which was stored in the IBM PC's secondary storage.

5.3. SYSTEM TESTING

The JSP methodology used for this research project made sure that software would have testability and maintainability inherent in the design. The steps of design included:

- the functional requirements of each part of the system to be listed,
- the performance requirements of each part of the system to be listed,
- detailed descriptions of interfaces,
- data types to be defined and
- test cases showing data to be input and expected results.

The functional requirements and performance requirements enabled black box testing of the major components of the system and software design. The detailed description of interfaces and data types enabled white box testing of individual processes to be tested thoroughly.

Each major component of the TMS; namely, the TMD, the DTD and the DPS were tested separately. Then the appropriate serial connections, between the major components, were made and the system was tested as a whole.

5.4. SUGGESTIONS FOR FURTHER CONSIDERATIONS

The temperature monitoring system was designed and tested in order to meet the original project proposal. It has been proved that a miniature data logging device could be manufactured and a support system could be implemented to a commercial standard. Future considerations include:

A future project could be to research into fabrication techniques that would enable a miniature logging system to be manufactured at a low cost. This would involve more time being spent on analysing the power requirements for such a device and how often the batteries would need to be replaced.

It is obvious that there are many similar devices that could use parts of this design so that other physiological parameters can be monitored, such as: blood pressure, heart beat rates and e.c.g. values.

Once a device has been fabricated, then the medical profession would have to research into the most suitable body sites where a device can be worn without causing any discomfort or inconvenience to the wearer.

It should be noted, that the author's responsibility was to provide the raw data. The medical profession would have to decide how the logged data can best be presented on a VDU screen and how information from a logging session should be arranged in order to produce a hard copy of the results.

Whilst gaining valuable experience with 8-bit microcontroller devices I realised the importance of these devices for future single-chip solutions to many of today's and future problems. There is a need to know how these microcontroller devices can be used in parallel, to form a bus system of devices or even a network of devices.

There is also an immediate need for engineers to be familiar with the latest 32-bit microcontroller devices, as they have the power to process the logged data and present it, in graphical form, to high resolution flat screen displays.

Finally, I believe that this project has solved the problem at hand but, in doing so, it has opened up a whole new exciting area of design. I hope to continue in the microcontroller applications design field and, in particular, research into the networking of microcontroller devices.

Figures Used Throughout This Document

FIGURE 1. CIRCADIAN RHYTHMS DIAGRAM

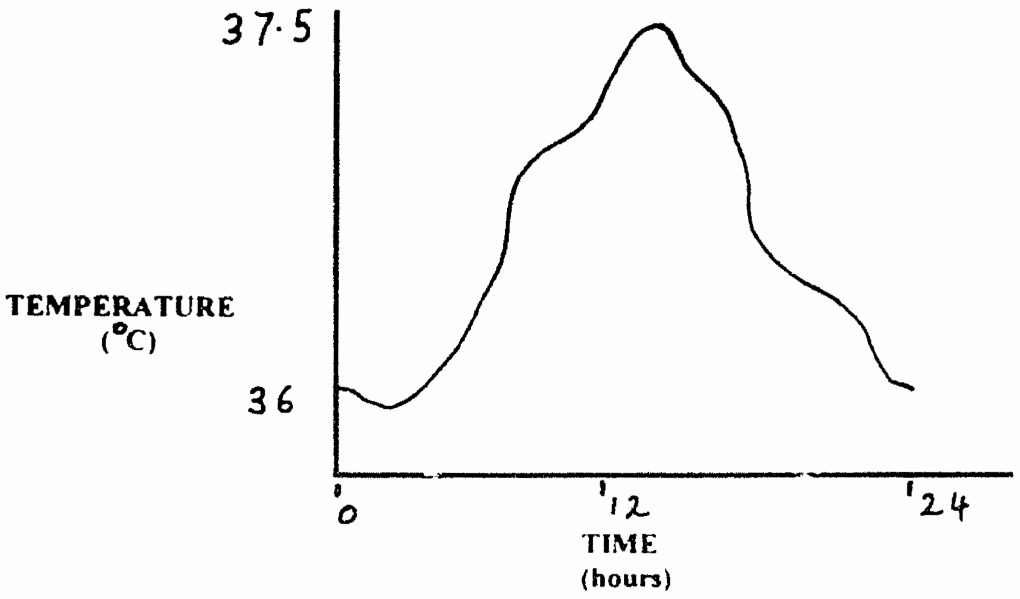


FIGURE 2. THERMISTOR'S THERMAL RESPONSE CHARACTERISTIC

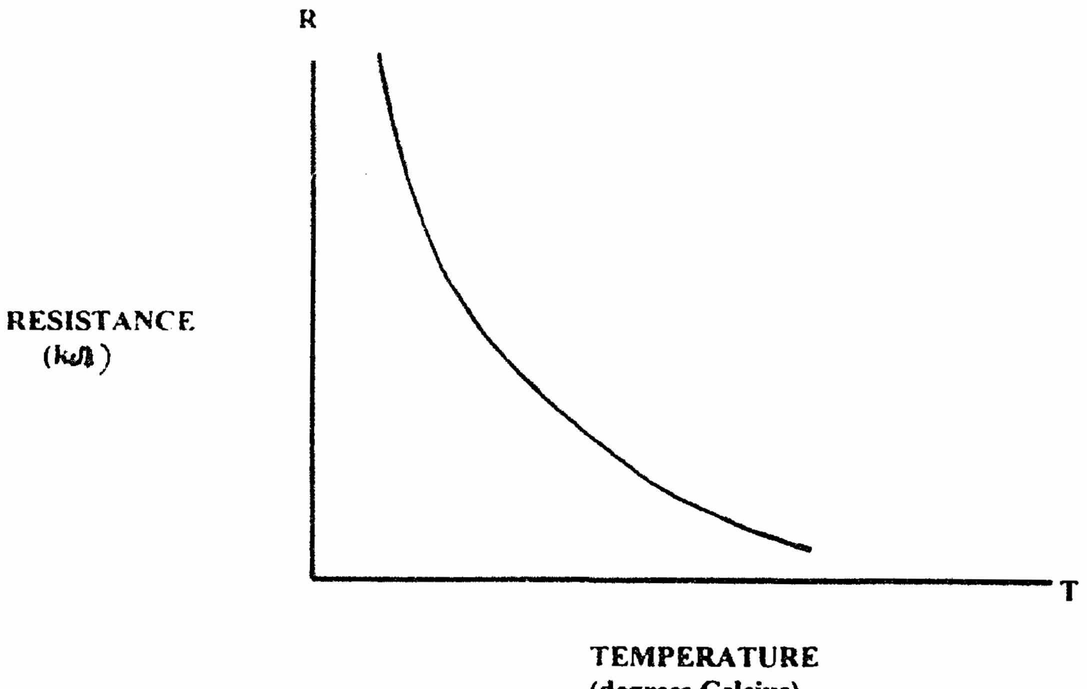


FIGURE 3.1. TEMPERATURE SENSOR CIRCUIT DIAGRAM

(Thermistor Sensor Circuits)

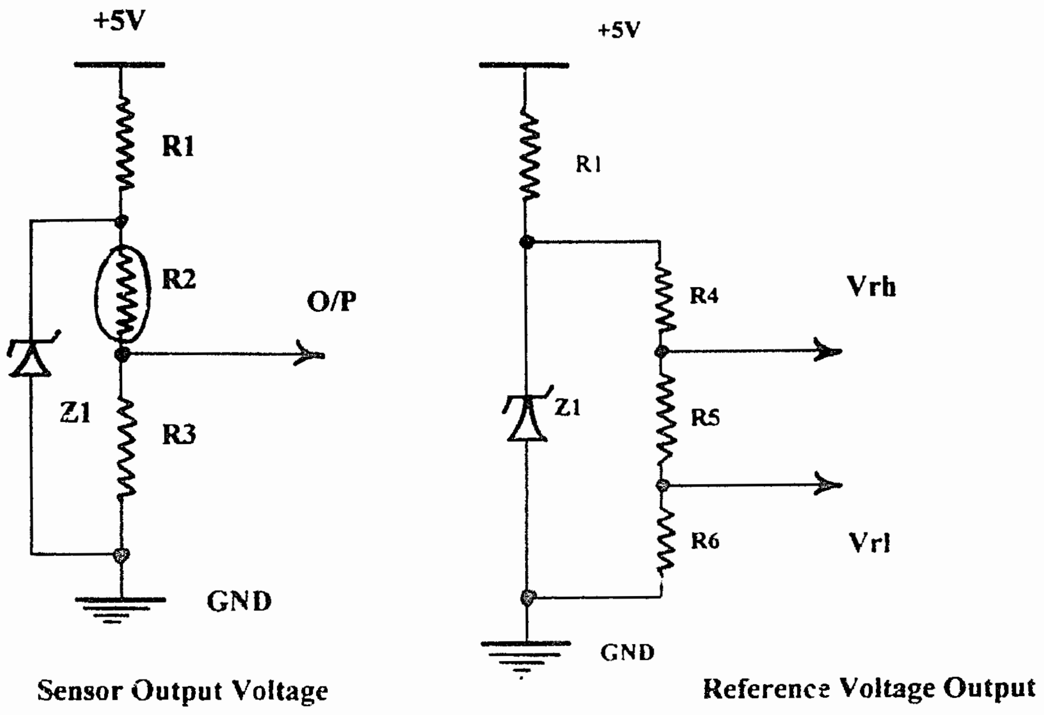


FIGURE 3.2. TEMPERATURE SENSOR CIRCUIT DIAGRAM
(IC Temperature Sensor Circuit)

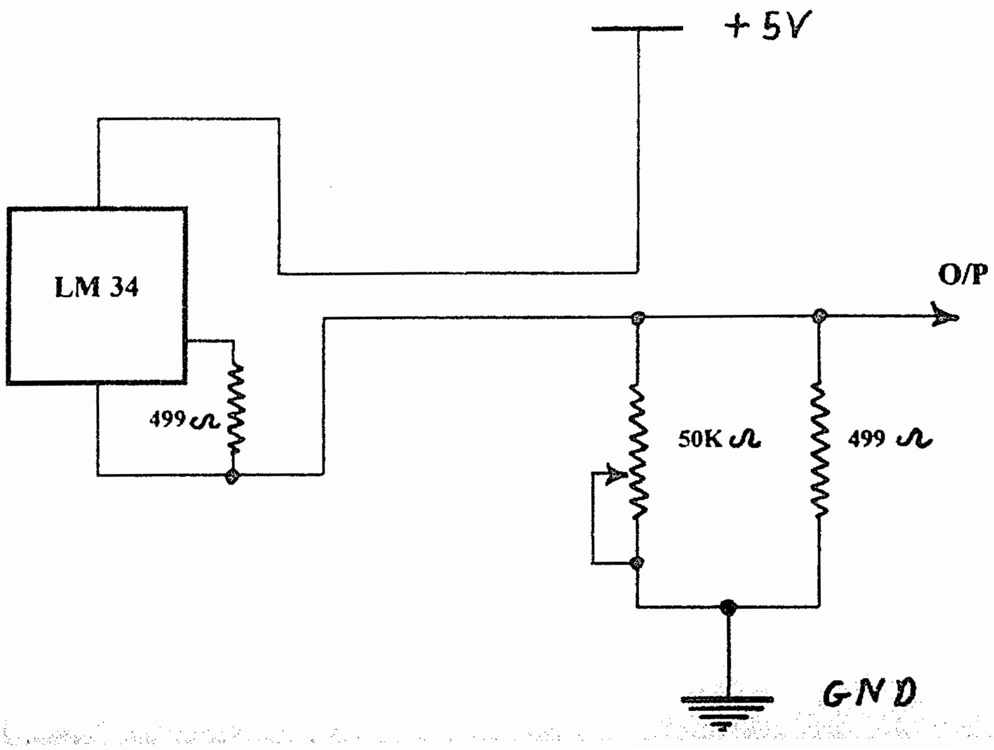
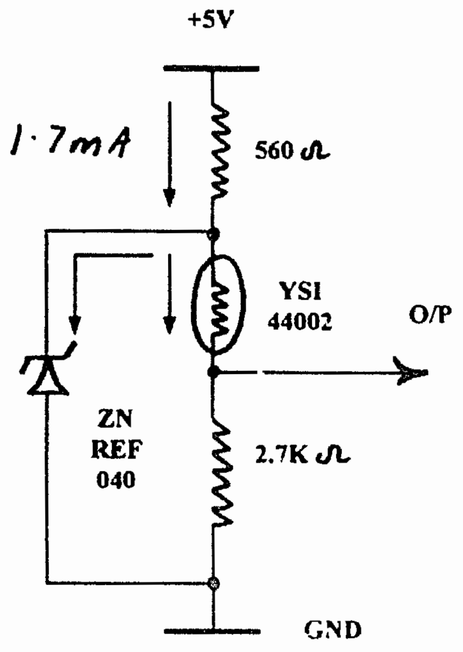
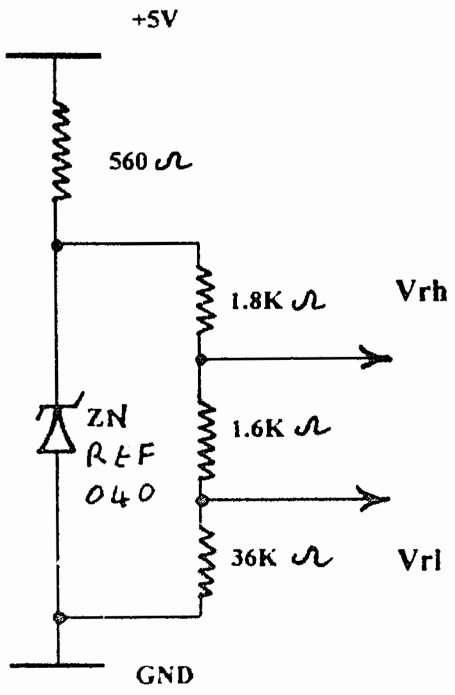


FIGURE 3.3. TEMPERATURE SENSOR CIRCUIT DIAGRAM

(Thermistor Sensor Circuits)



Sensor Output Voltage



Reference Voltage Outputs

FIGURE 4 THE MC68HC11 BLOCK DIAGRAM

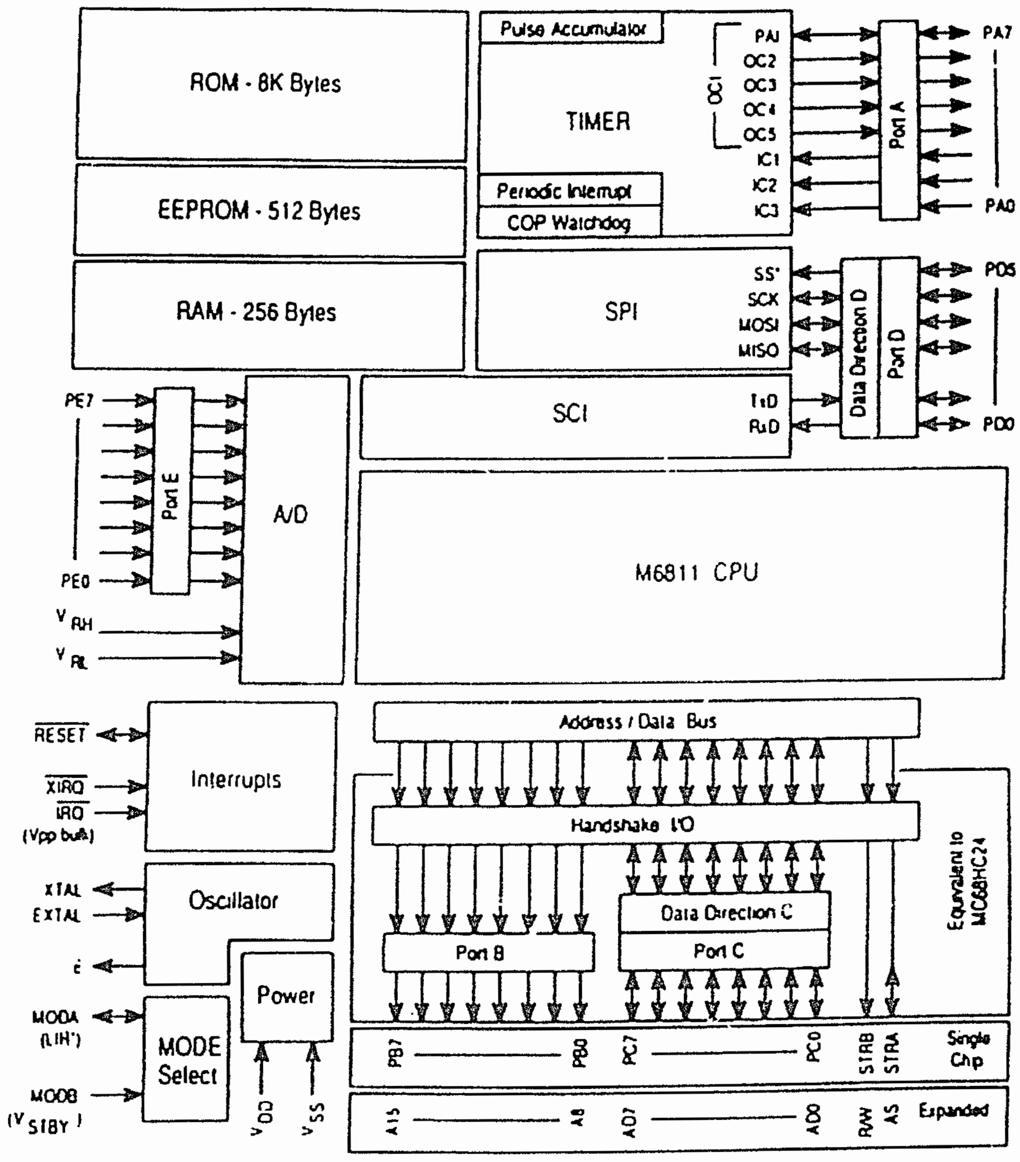


FIGURE 5

MEMORY MAP OF THE MC6811 SYSTEM

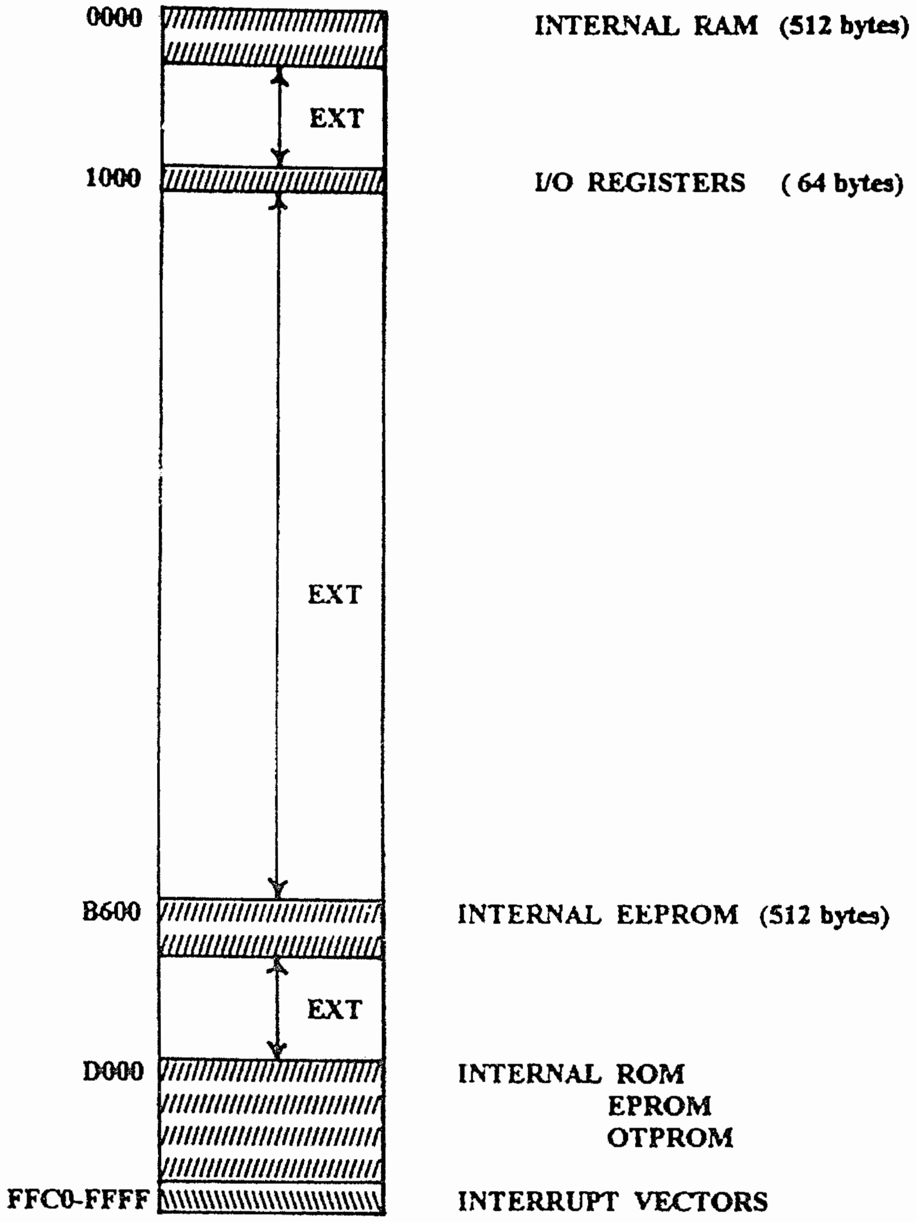


FIGURE 6 **MC68HC11 DESIGN AND TEST ENVIRONMENT**

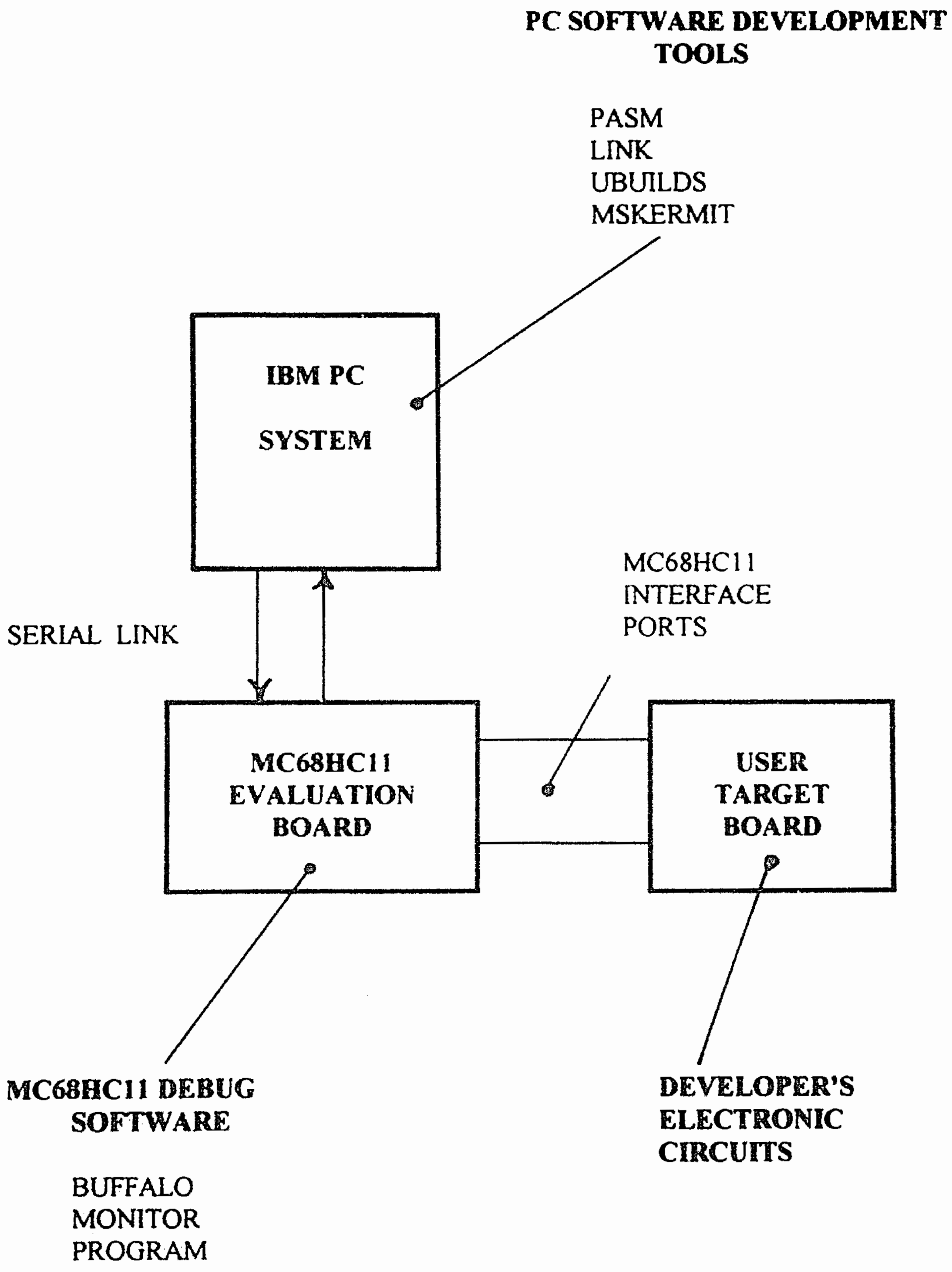


FIGURE 7. TEMPERATURE MONITORING SYSTEM
(Block Diagram)

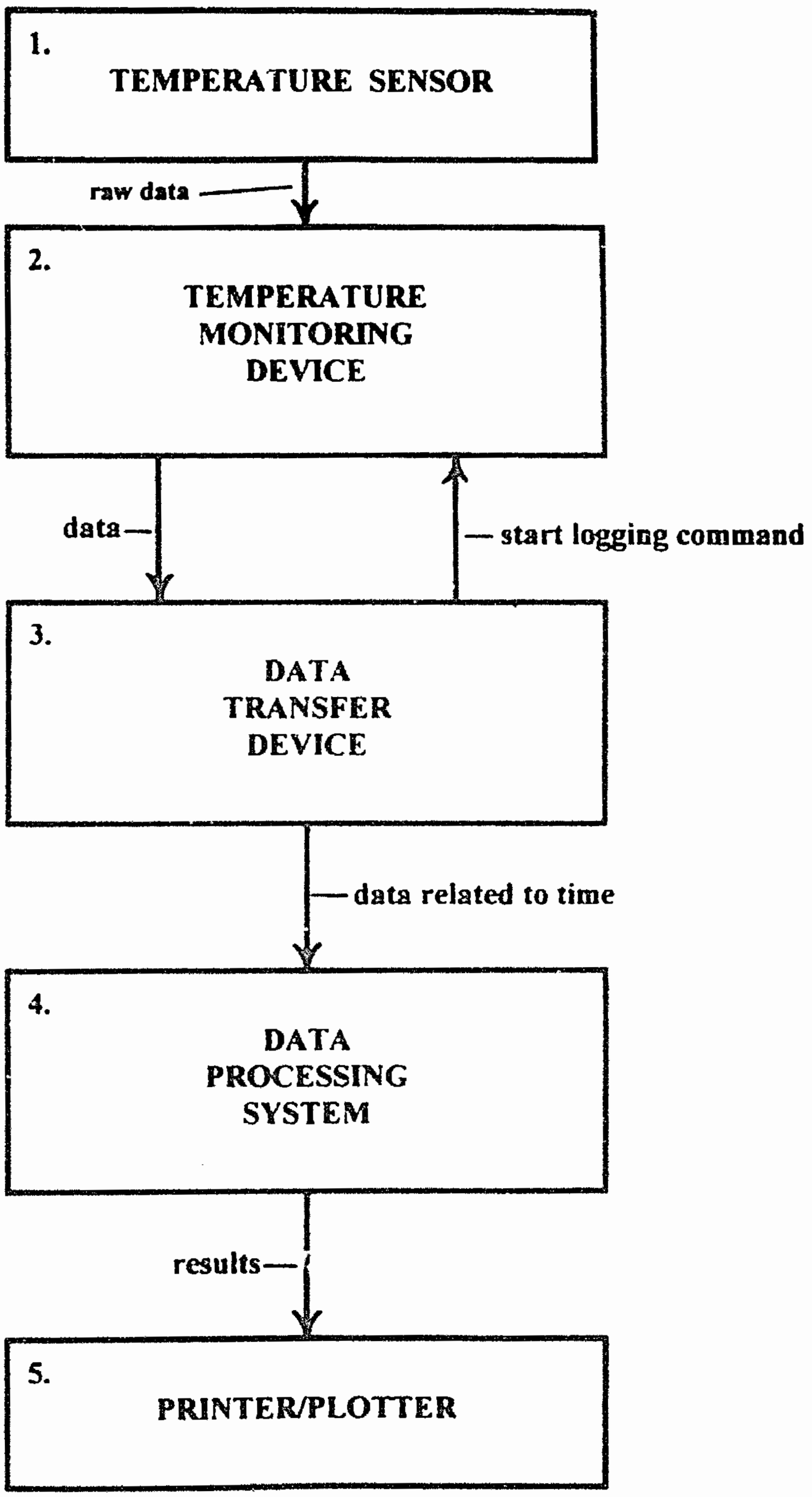


FIGURE 8 THE TEMPERATURE MONITORING DEVICE
(Block Diagram)

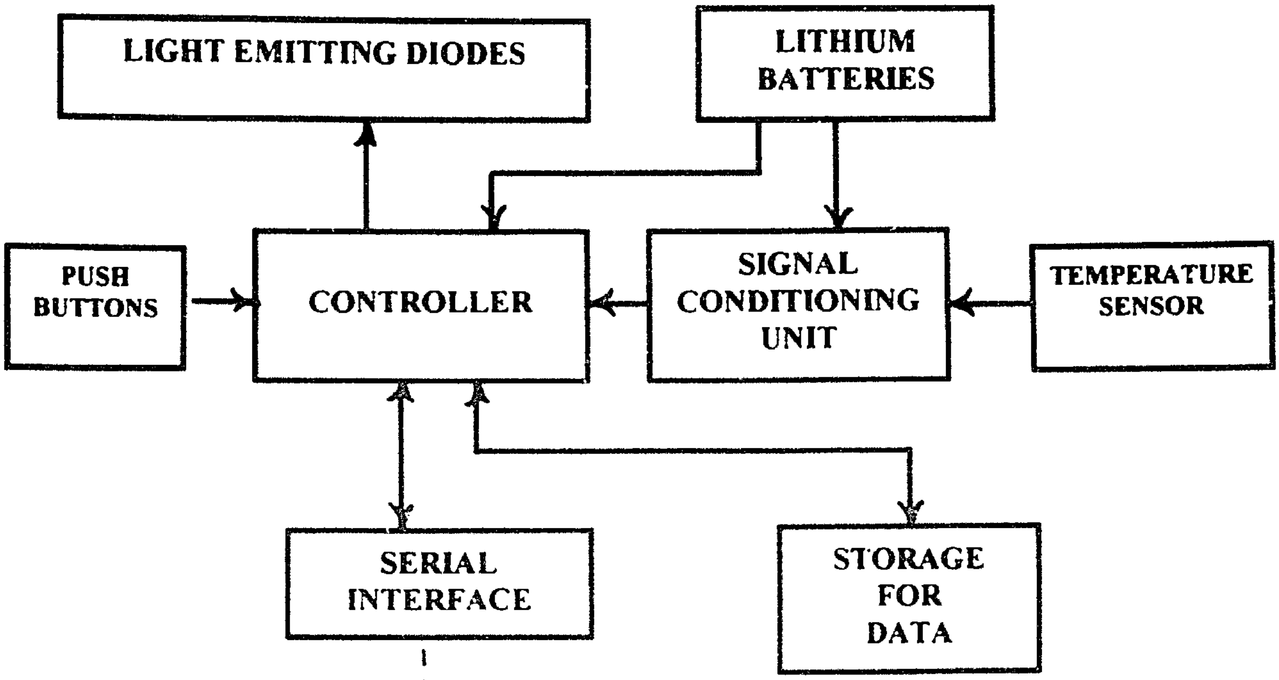


FIGURE 9 DATA TRANSFER DEVICE
(Block Diagram)

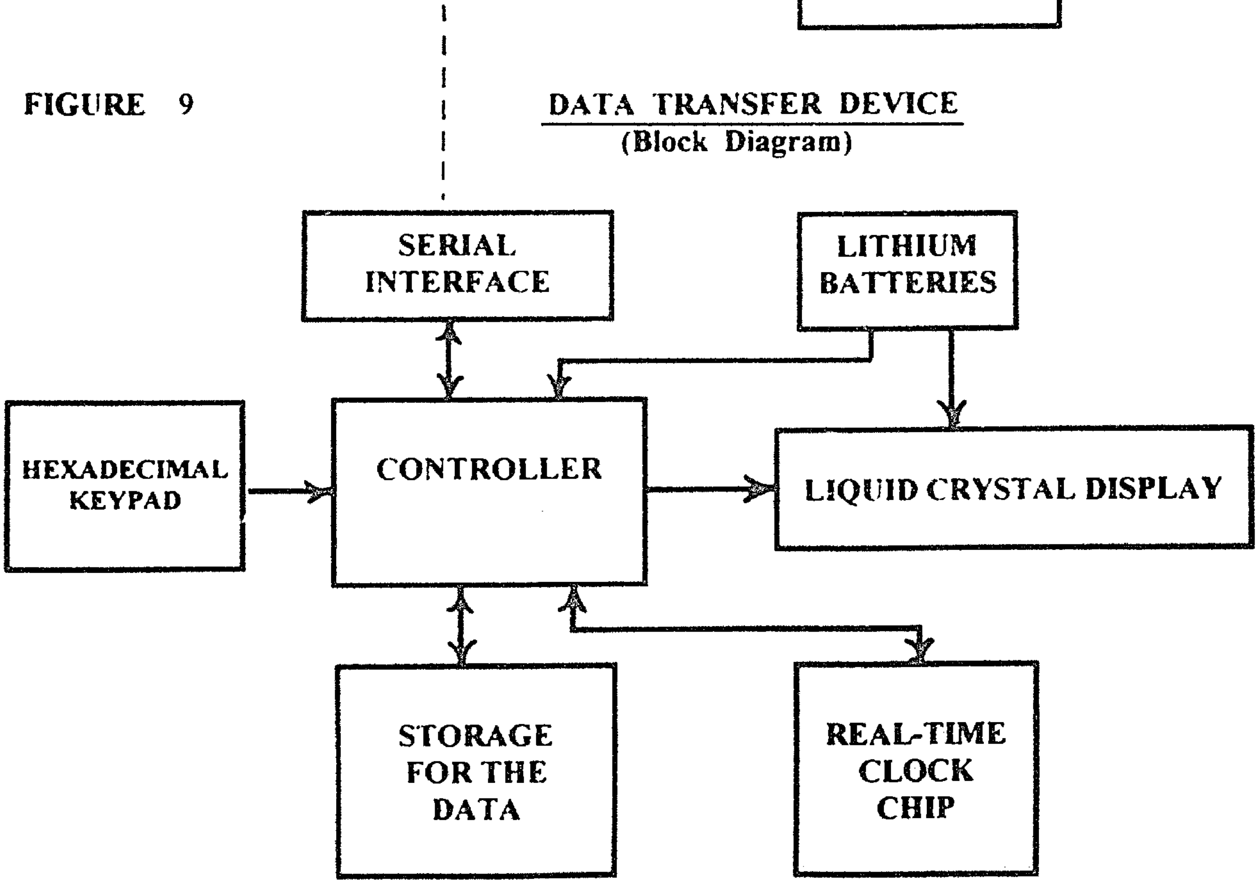


FIGURE 10. THE DATA PROCESSING SYSTEM
(Block Diagram)

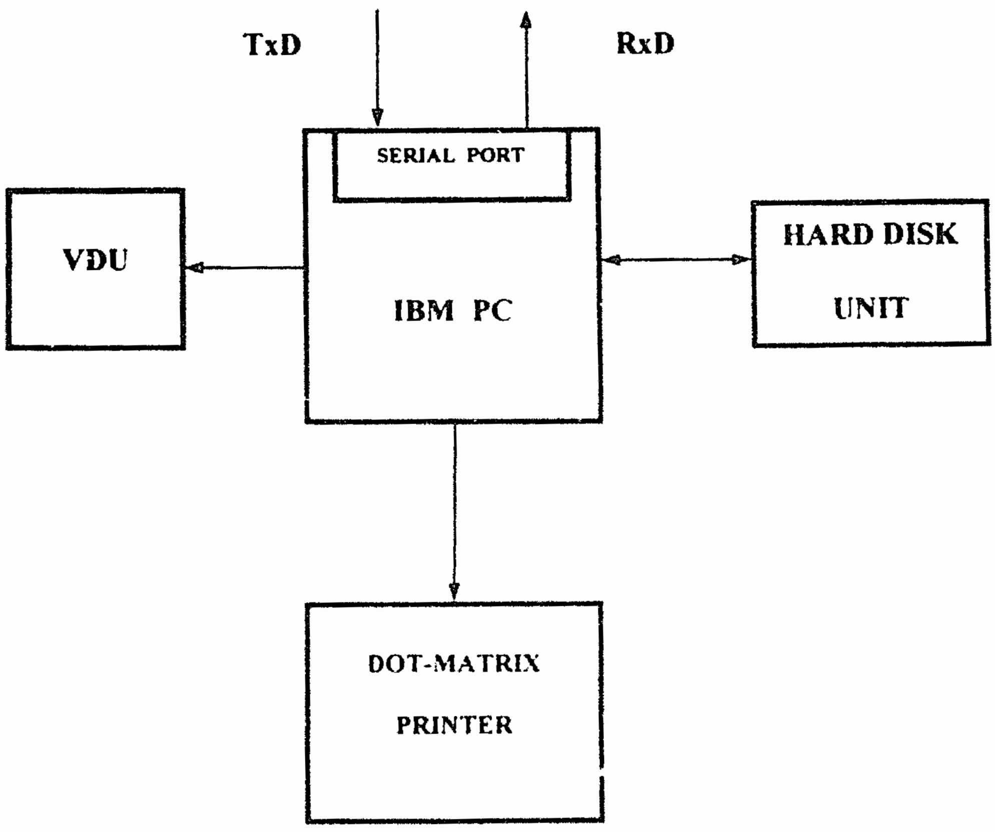


FIGURE 11. THE SYSTEM MODEL DIAGRAM

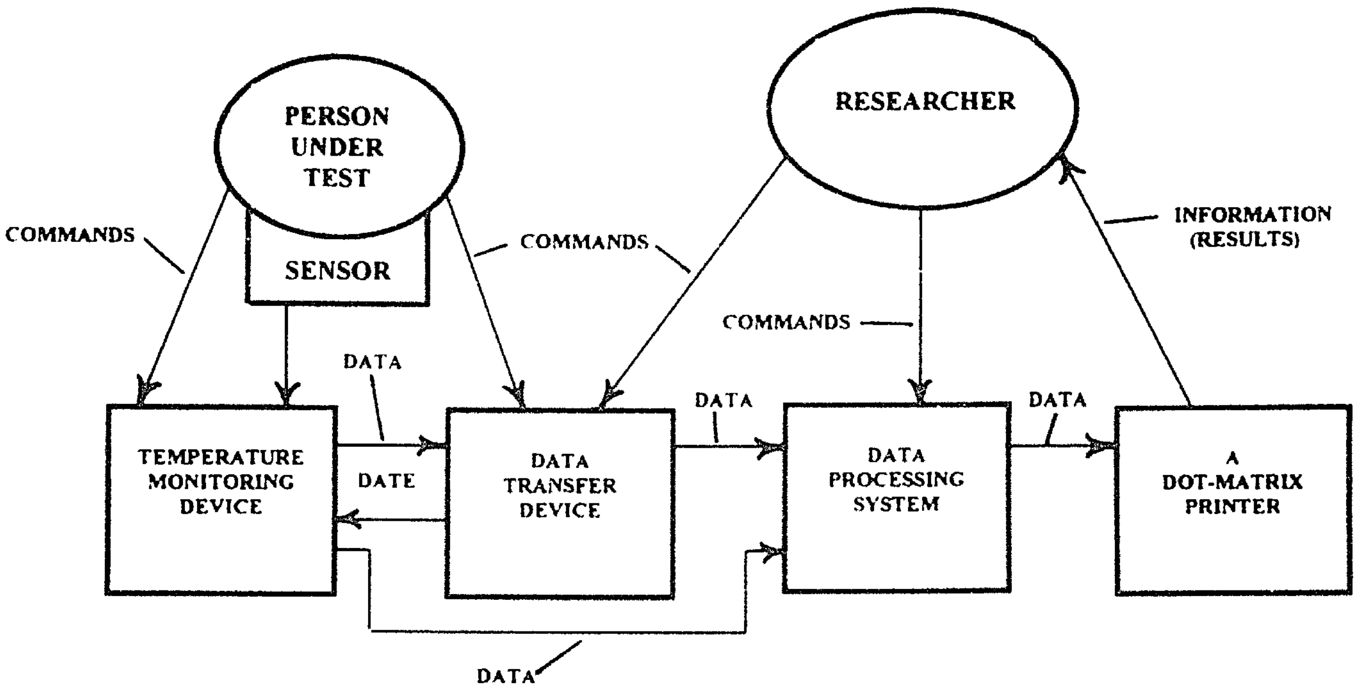


FIGURE 12. TEMPERATURE MONITORING DEVICE

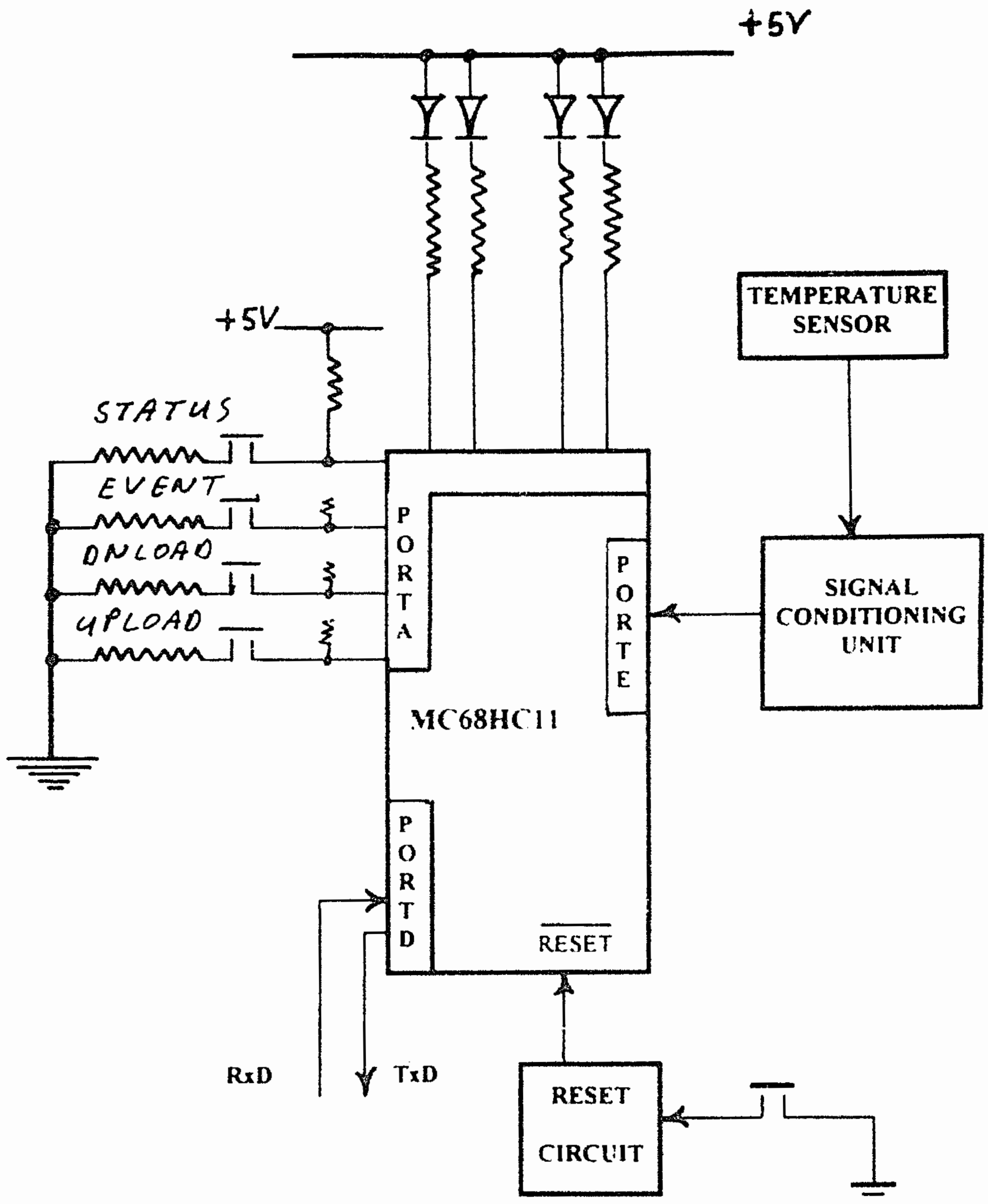
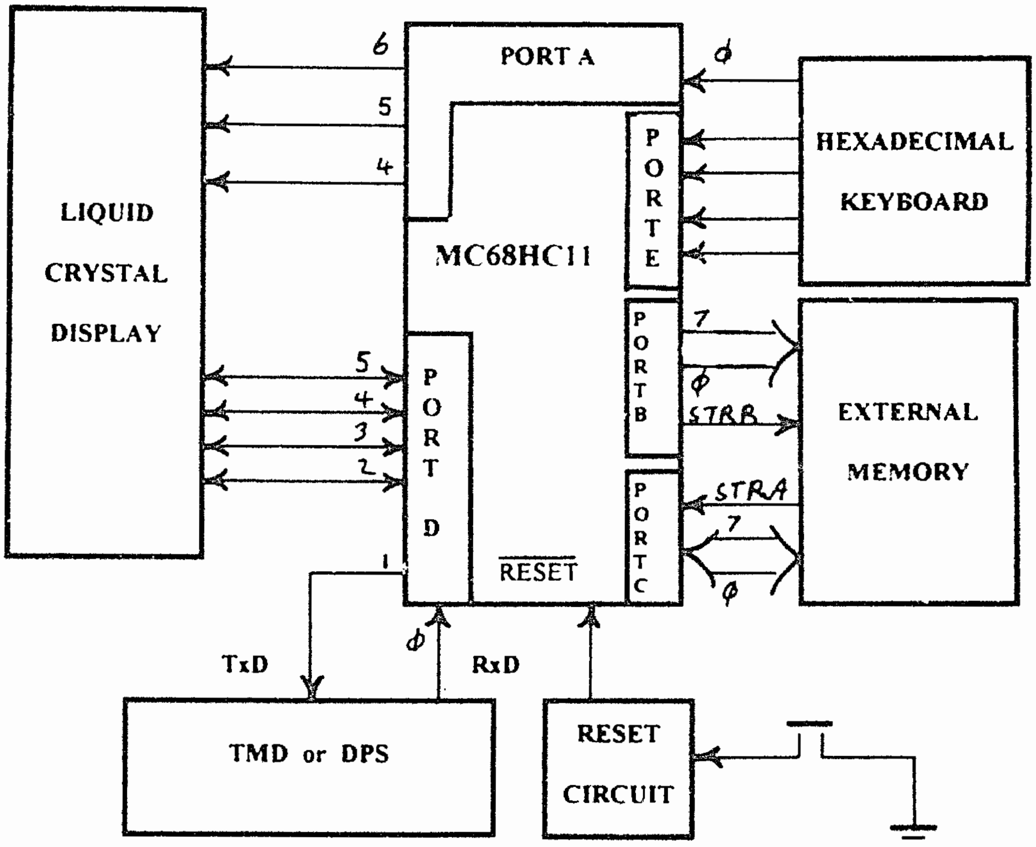


FIGURE 13. DATA TRANSFER DEVICE



Volume 2

APPENDICES

APPENDIX A

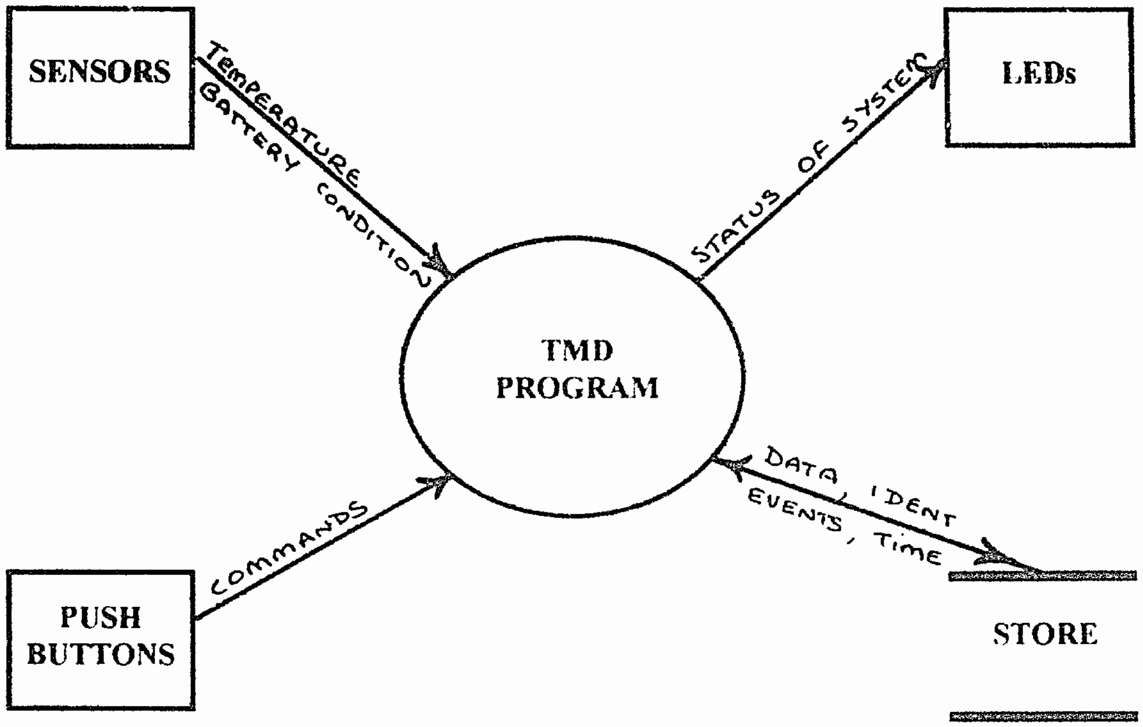
The Temperature Monitor Device

This section contains data flow diagrams, Jackson structure diagrams and program listing for the temperature monitoring device.

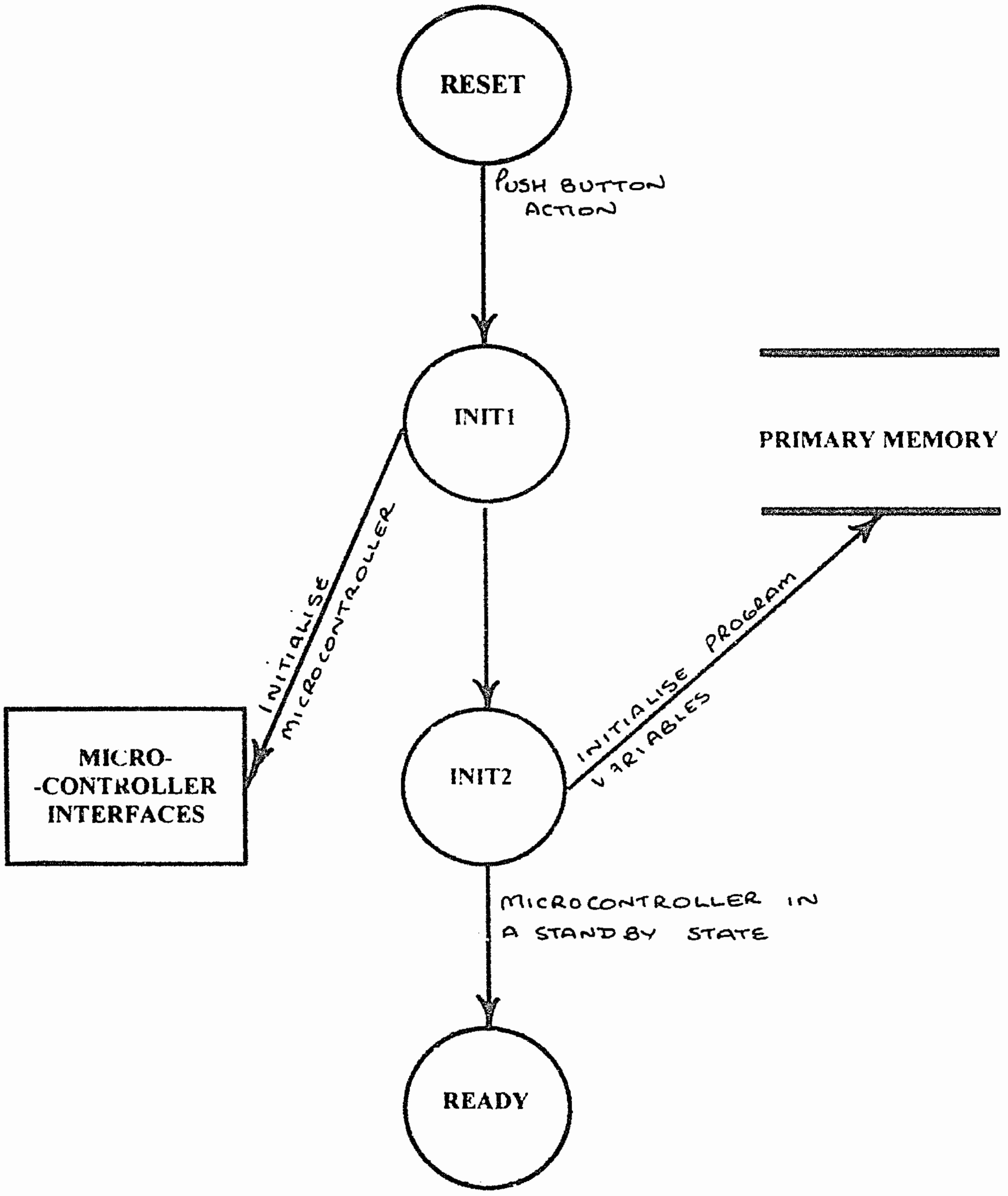
A.1.

The Temperature Monitor Device data flow diagrams

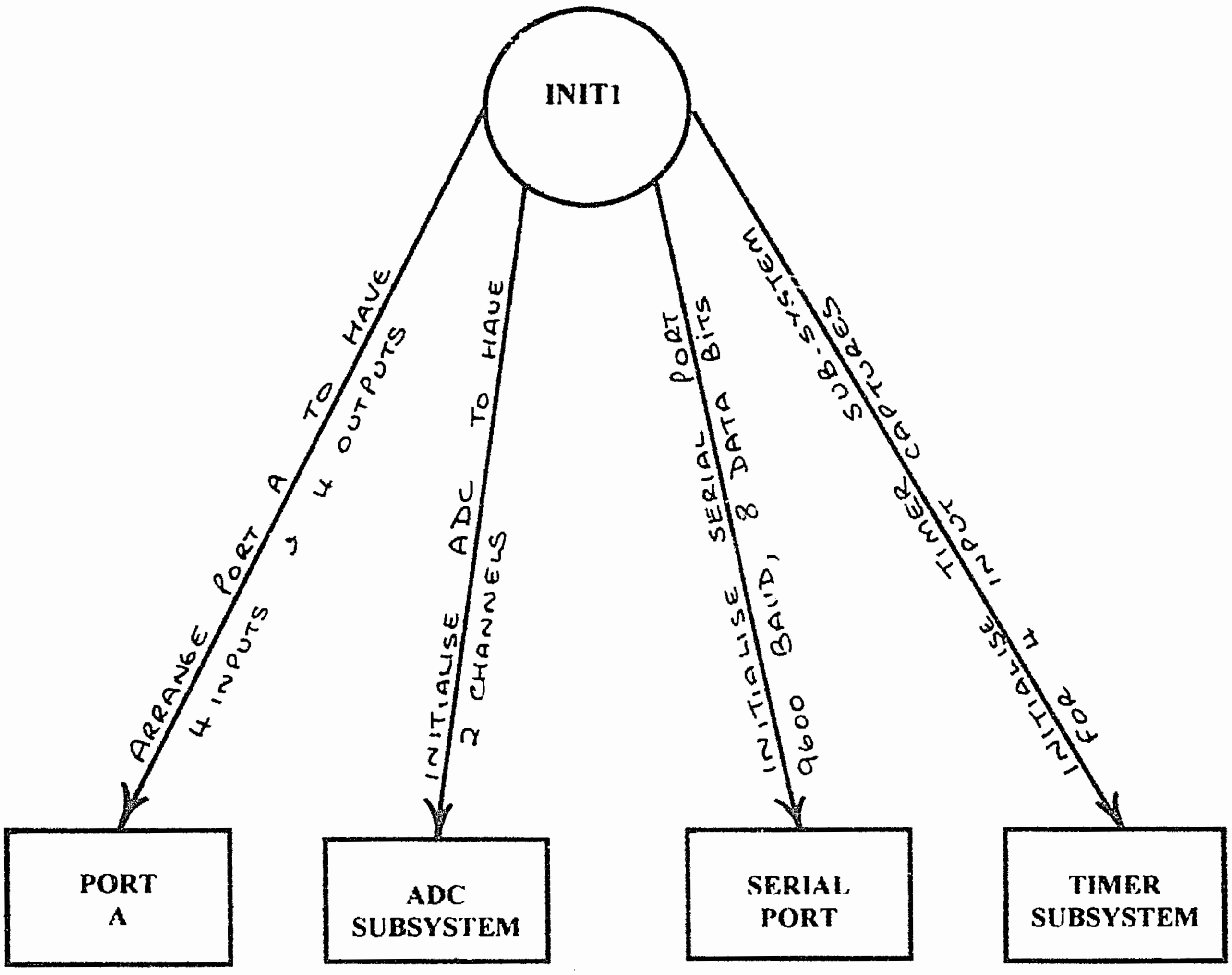
DFD LEVEL 0


SENSORS
TEMPERATURE
BATTERY CONDITION
LEDs
STATUS OF SYSTEM
TMD PROGRAM
PUSH BUTTONS
COMMANDS
DATA, IDENT
EVENTS, TIME
STORE

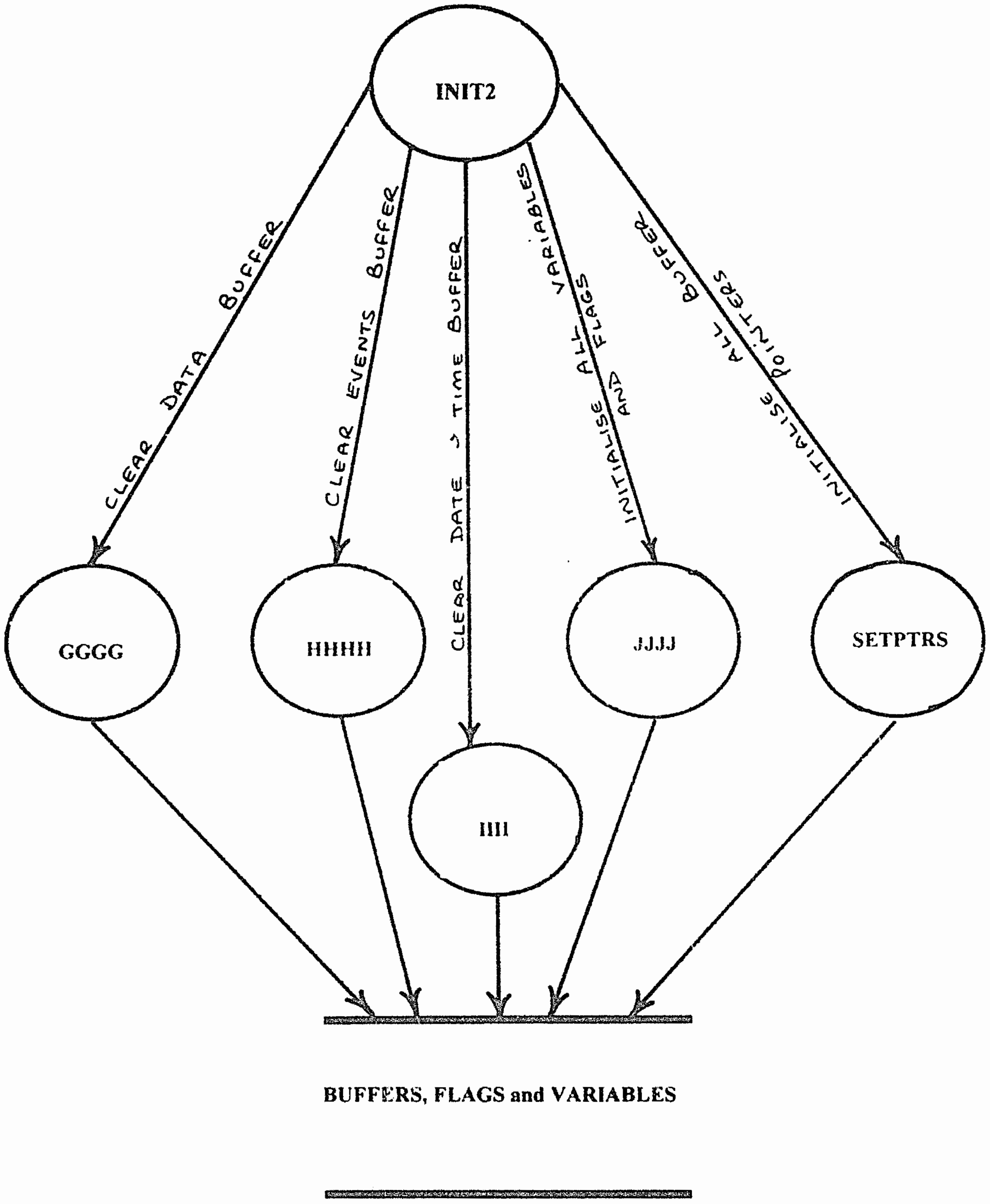
DFD LEVEL 1


RESET
PUSH BUTTON ACTION
INIT1
PRIMARY MEMORY
INITIALISE MICROCONTROLLER
MICRO- -CONTROLLER INTERFACES
INIT2
INITIALISE PROGRAM VARIABLES
MICROCONTROLLER IN A STANDBY STATE
READY

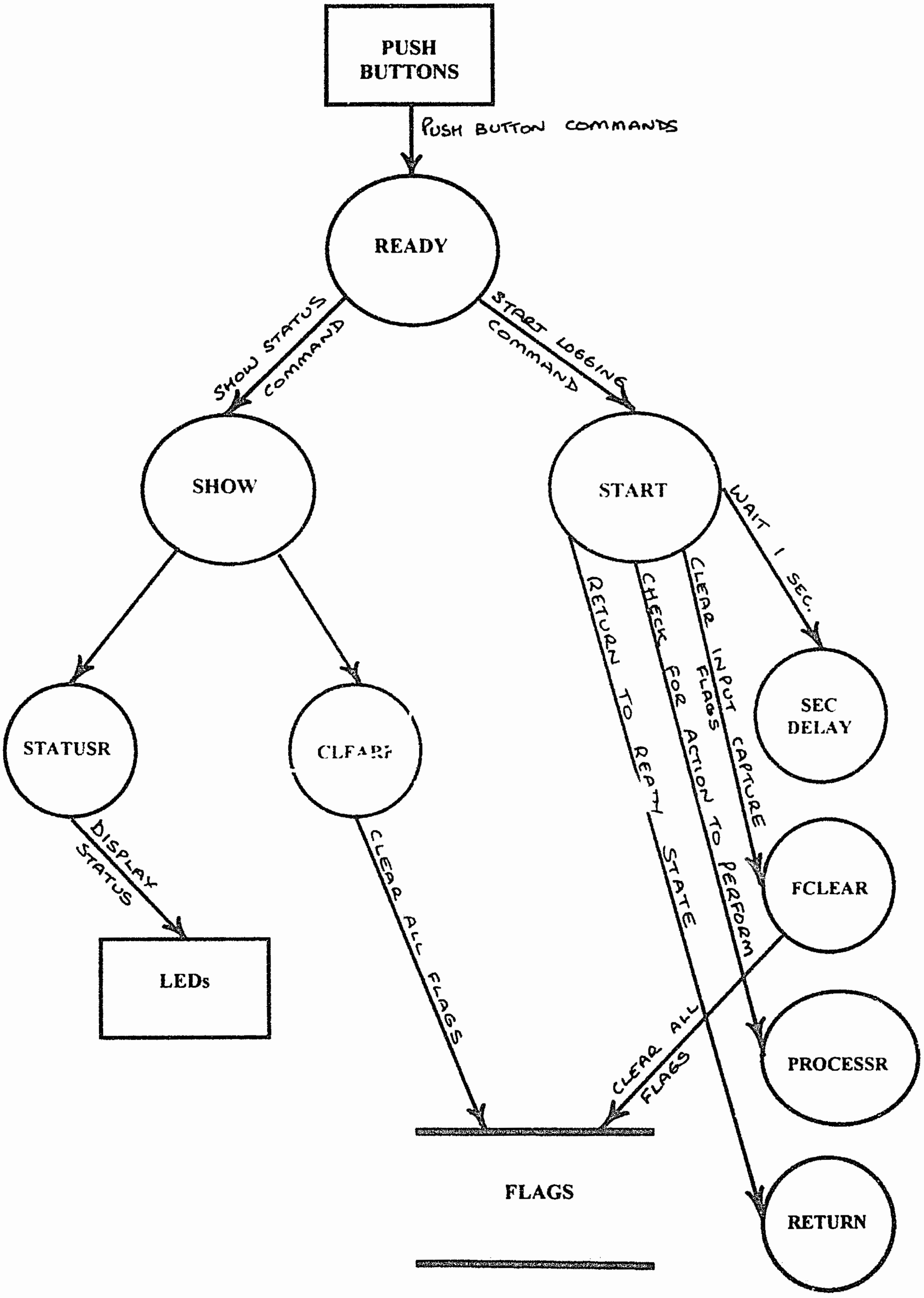
DFD LEVEL 2


INIT1
ARRANGE PORT A TO HAVE 4 INPUTS & 4 OUTPUTS
INITIALISE ADC TO HAVE 2 CHANNELS
INITIALISE SERIAL PORT 8 DATA BITS 9600 BAUD,
INITIALISE TIMER SUB-SYSTEM FOR 4 INPUT CAPTURES
PORT A
ADC SUBSYSTEM
SERIAL PORT
TIMER SUBSYSTEM

DFD LEVEL 2

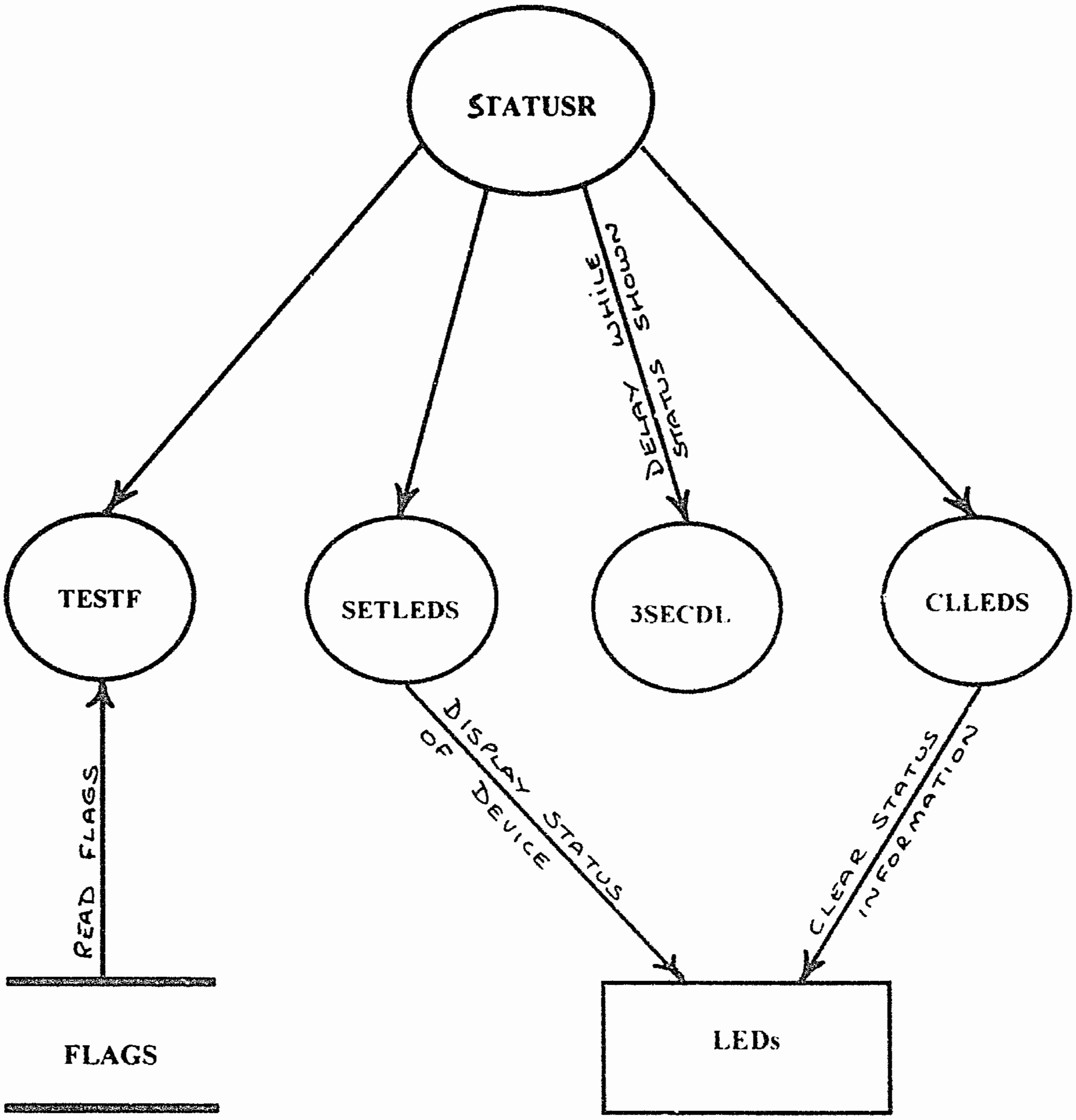

INIT2
CLEAR DATA BUFFER
CLEAR EVENTS BUFFER
CLEAR DATE & TIME BUFFER
INITIALISE ALL VARIABLES AND FLAGS
INITIALISE ALL BUFFER POINTERS
GGGG
HHHHH
IIII
JJJJ
SETPTRS
BUFFERS, FLAGS and VARIABLES

DFD LEVEL 2

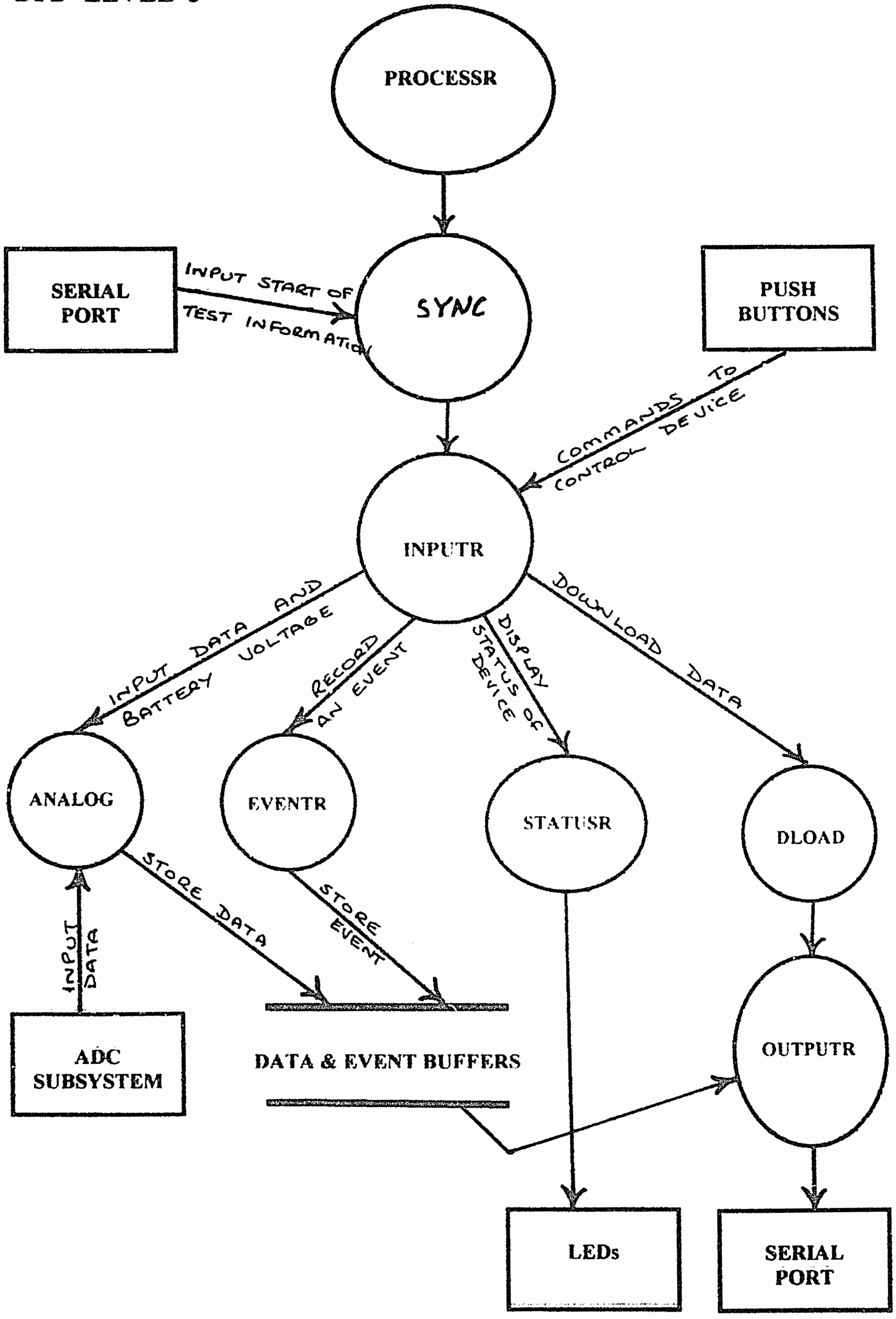


PUSH BUTTONS
PUSH BUTTON COMMANDS
READY
SHOW STATUS COMMAND
START LOGGING COMMAND
SHOW
START
WAIT 1 SEC.
RETURN TO READY STATE
CHECK FOR ACTION TO PERFORM
CLEAR INPUT CAPTURE FLAGS
STATUSR
CLEARF
SEC DELAY
FCLEAR
DISPLAY STATUS
CLEAR ALL FLAGS
CLEAR ALL FLAGS
LEDs
PROCESSR
FLAGS
RETURN

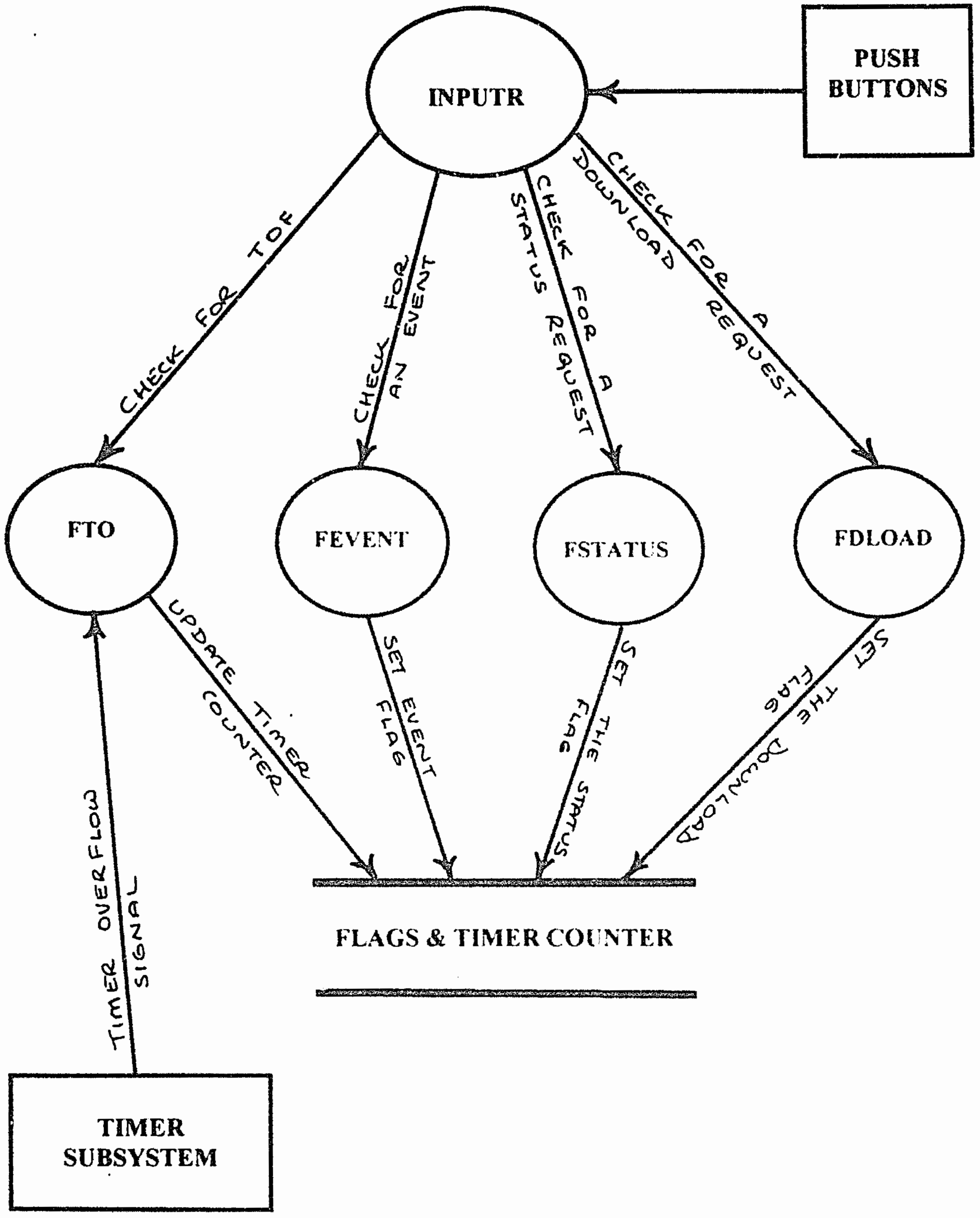
DFD LEVEL 3


STATUSR
TESTF
SETLEDS
3SECDL
CLLEDS
DELAY WHILE STATUS SHOWN
READ FLAGS
DISPLAY STATUS OF DEVICE
CLEAR STATUS INFORMATION
FLAGS
LEDs

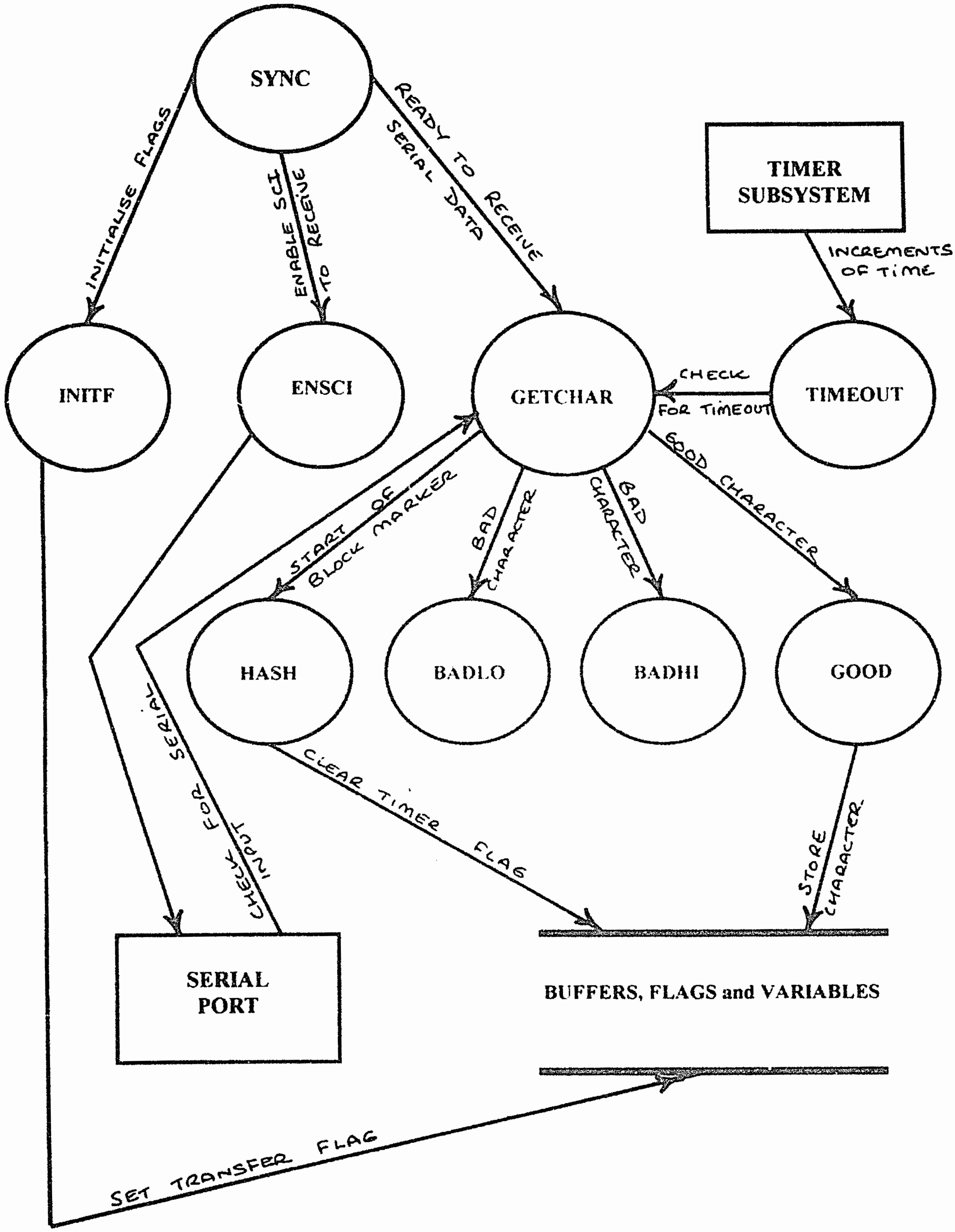
DFD LEVEL 3


PROCESSR
SERIAL PORT
INPUT START OF TEST INFORMATION
SYNC
PUSH BUTTONS
COMMANDS TO CONTROL DEVICE
INPUTR
INPUT DATA AND BATTERY VOLTAGE
RECORD AN EVENT
DISPLAY STATUS OF DEVICE
DOWNLOAD DATA
ANALOG
EVENTR
STATUSR
DLOAD
STORE DATA
STORE EVENT
INPUT DATA
ADC SUBSYSTEM
DATA & EVENT BUFFERS
OUTPUTR
LEDs
SERIAL PORT

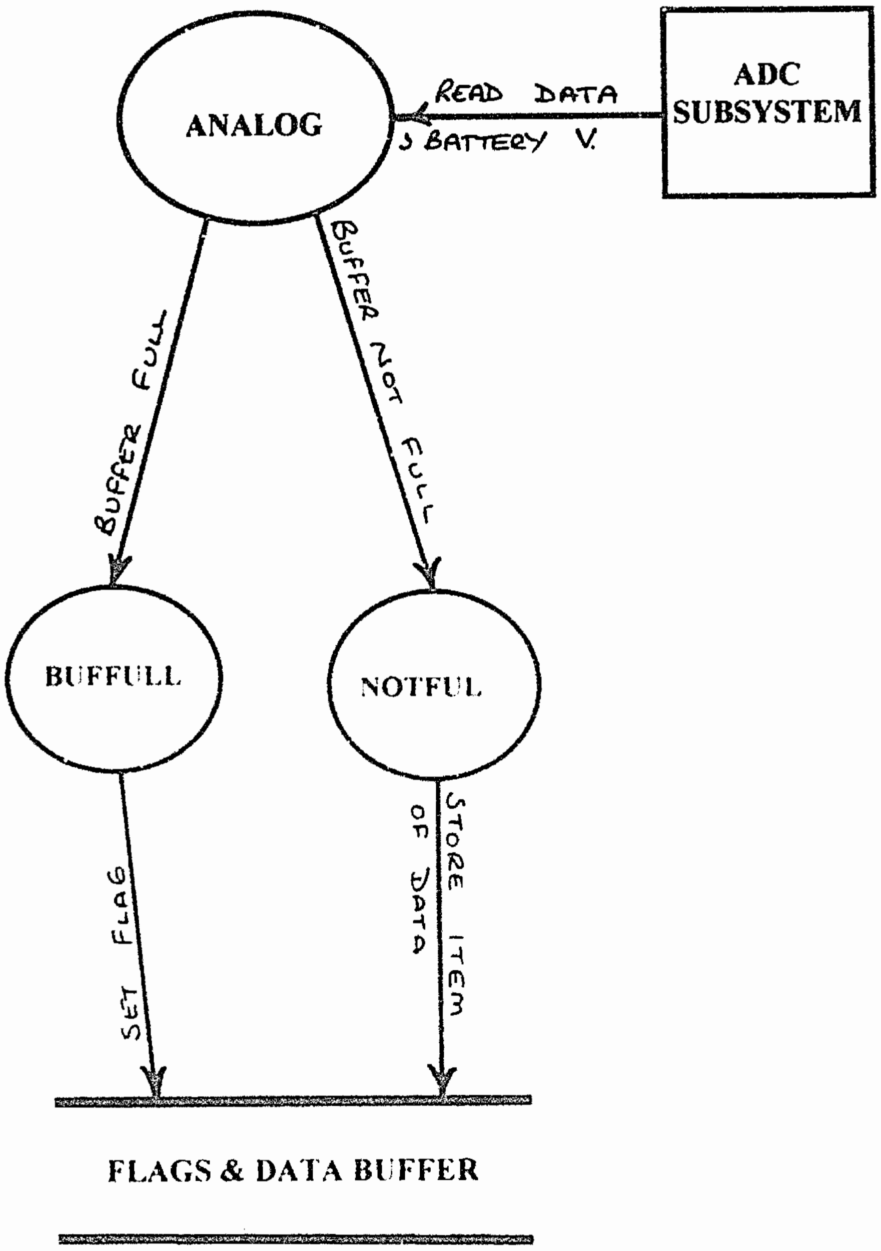
DFD LEVEL 4


PUSH BUTTONS
INPUTR
CHECK FOR TOF
CHECK FOR AN EVENT
CHECK FOR A STATUS REQUEST
CHECK FOR A DOWNLOAD REQUEST
FTO
FEVENT
FSTATUS
FDLOAD
UPDATE TIMER COUNTER
SET EVENT FLAG
SET THE STATUS FLAG
SET THE DOWNLOAD FLAG
FLAGS & TIMER COUNTER
TIMER OVERFLOW SIGNAL
TIMER SUBSYSTEM

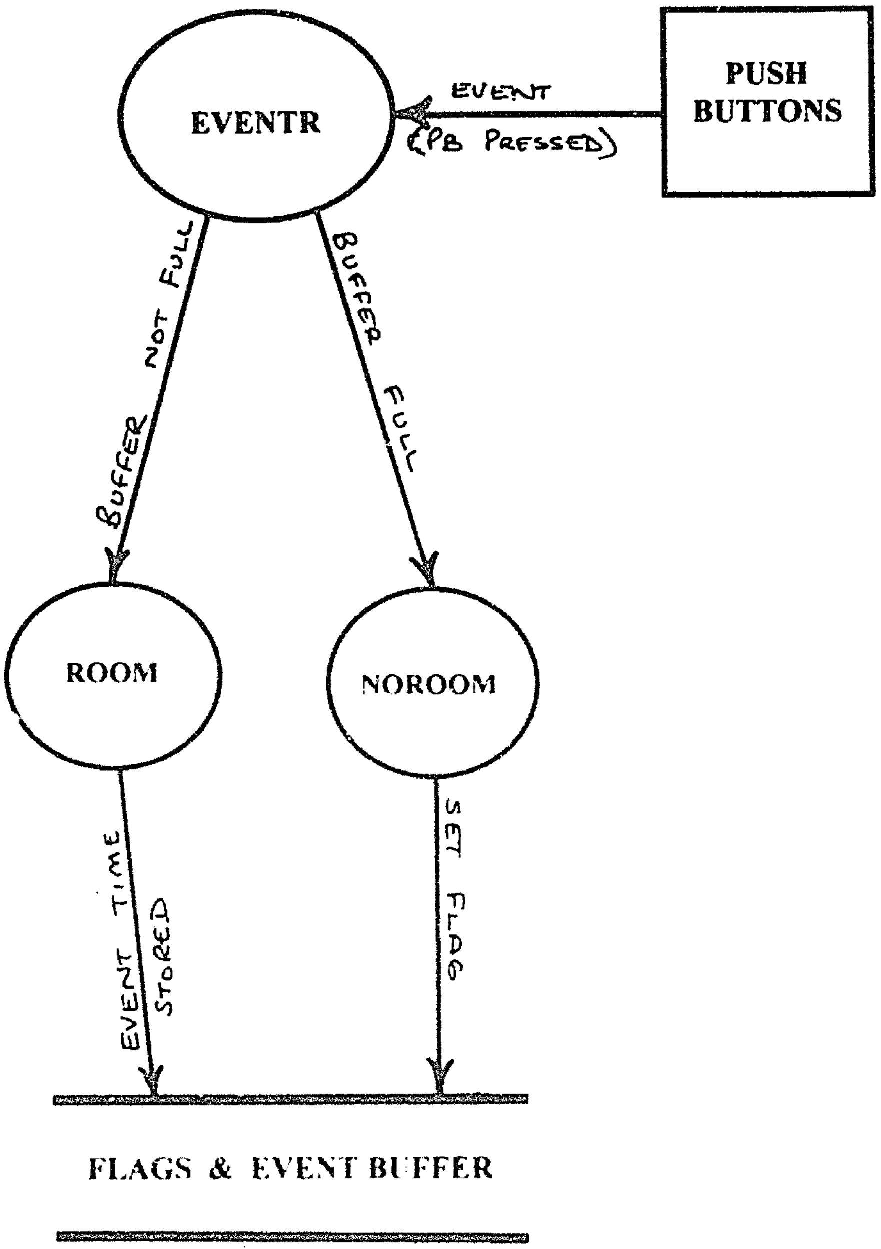
DFD LEVEL 4


SYNC
INITIALISE FLAGS
ENABLE SCI TO RECEIVE
READY TO RECEIVE SERIAL DATA
TIMER SUBSYSTEM
INCREMENTS OF TIME
INITF
ENSCI
GETCHAR
TIMEOUT
CHECK FOR TIMEOUT
START OF BLOCK MARKER
BAD CHARACTER
BAD CHARACTER
GOOD CHARACTER
HASH
BADLO
BADHI
GOOD
CHECK FOR SERIAL INPUT
CLEAR TIMER FLAG
STORE CHARACTER
SERIAL PORT
BUFFERS, FLAGS and VARIABLES
SET TRANSFER FLAG

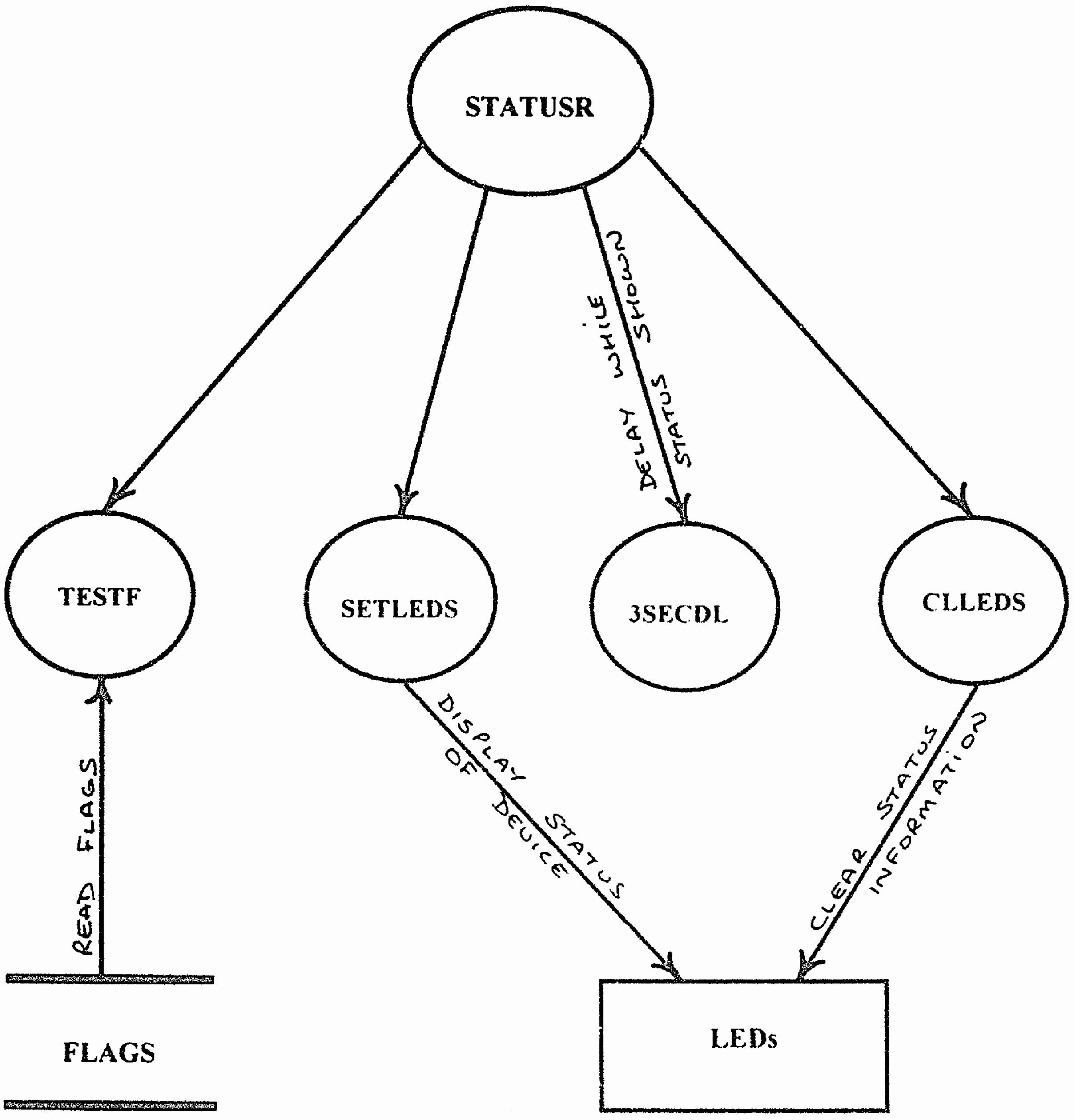
DFD LEVEL 5


ANALOG
ADC SUBSYSTEM
READ DATA
& BATTERY V.
BUFFER FULL
BUFFER NOT FULL
BUFFULL
NOTFUL
SET FLAG
STORE ITEM OF DATA
FLAGS & DATA BUFFER

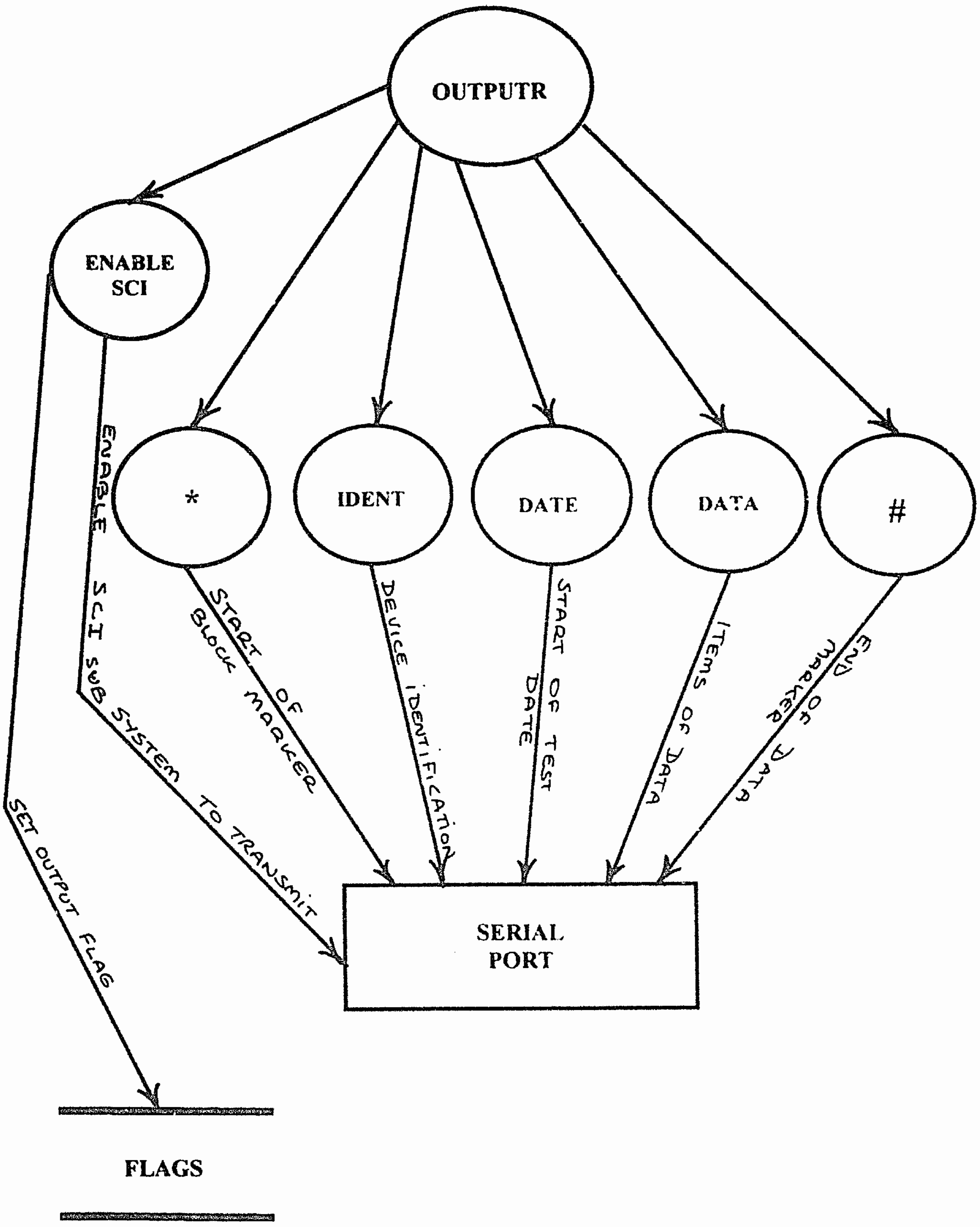
DFD LEVEL 5


PUSH BUTTONS
EVENT (PB PRESSED)
EVENTR
BUFFER NOT FULL
BUFFER FULL
ROOM
NOROOM
EVENT TIME STORED
SET FLAG
FLAGS & EVENT BUFFER

DFD LEVEL 5


STATUSR
TESTF
SETLEDS
3SECDL
CLLEDS
DELAY WHILE STATUS SHOWN
READ FLAGS
FLAGS
DISPLAY STATUS OF DEVICE
CLEAR STATUS INFORMATION
LEDs

DFD LEVEL 5

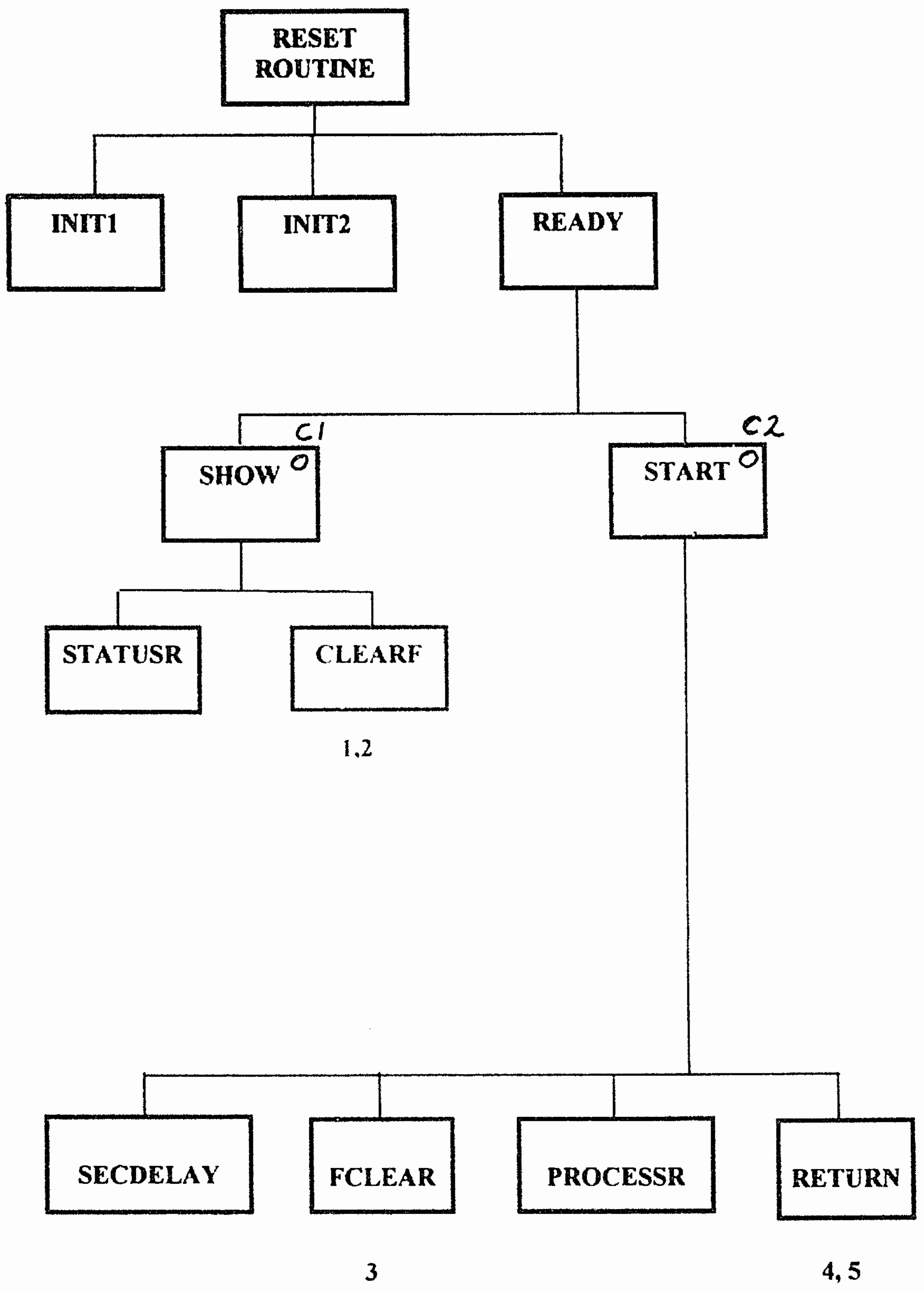

OUTPUTR
ENABLE SCI
*
IDENT
DATE
DATA
#
ENABLE SCI SUB SYSTEM TO TRANSMIT
START OF BLOCK MARKER
DEVICE IDENTIFICATION
START OF TEST DATE
ITEMS OF DATA
END OF MARKER DATA
SET OUTPUT FLAG
SERIAL PORT
FLAGS

A.2.

The Temperature Monitor Device

Jackson structure diagrams

TMD PROGRAM STRUCTURE



CONDITIONS:

C1. Selected when the `status' pushbutton has been pressed.

C2. Selected when the `upload/download' pushbutton has been pressed.

FUNCTIONS:

1. Clear the input capture flag bits.
2. Get back to ready state.

3. Clear the input capture flags.

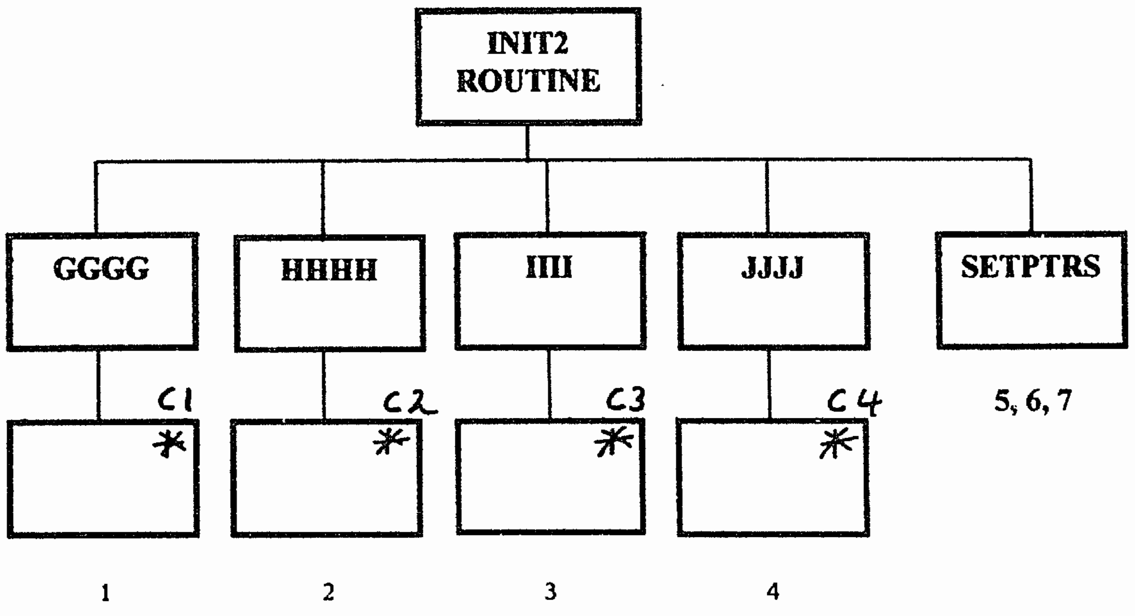
4. Clear the input capture flags.
5. Branch to READY for another command.

INIT1

FUNCTIONS:

1. Arrange PORTA for 4 i/ps and for 4 o/ps.
2. Initialise the input capture system so that it detects positive edge pulses.
3. Arrange PORTD for 8 i/ps.
4. Initialise the ADC Sub system.
5. Delay (100 uSec) to allow the ADC to initialise itself.

6. Set the latch, on the EVB board, to enable Receiver data (RxD) to appear at PD0.
7. Set the SCI Sub system baud rate to 9600.
8. Set the SCI for 8 data bits.
9. Disable the SCI transmitter/ receiver circuits. Disable the SCI hardware interrupts.
10. Disable TIMER interrupts to avoid output compare action.
11. Clear all the input capture flags.
12. Return to the RESET routine.



CONDITIONS:

C1.	Clear locations between	DATA	and	BUFFERMAX.
C2.	Clear locations between	EVENTS	and	PTREMAX.
C3.	Clear locations between	DATE	and	DATEMAX.
C4.	Clear locations between	CLEAR	and	VARYMAX.

FUNCTIONS:

1. Clear the data buffer.

2. Clear the events buffer.

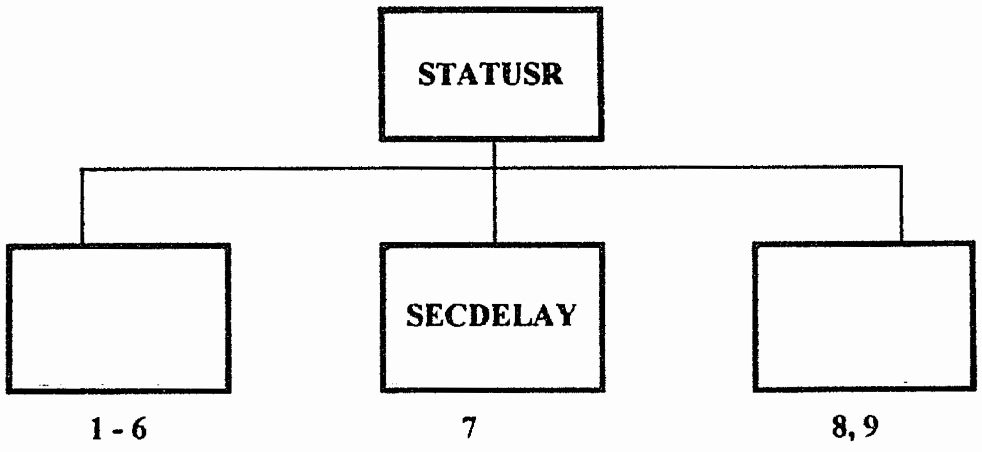
3. Clear the date and time buffer.

4. Clear all the variables and flags that are used by the TMD program.

5. Initialise the data buffer pointer to the start of the buffer.

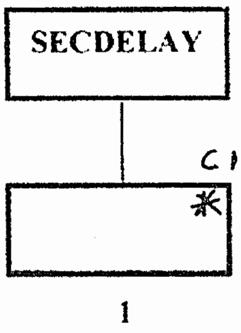
6. Initialise the events buffer pointer to the start of the buffer.

7. Return to the RESET routine.



FUNCTIONS:

1. Clear all the LED output signals.
2. Test the MONITOR status flag.
   IF set, the appropriate LED output bit of the signal has to be set.
3. Test the MEMFULL status flag.
   IF set, the appropriate LED output bit of the signal has to be set.
4. Test the BATTERY condition status flag.
   IF set, the appropriate LED output bit of the signal has to be set.
5. Test the TRANSFER of serial data status flag.
6. Output the system status to PORTA.
7. SECDELAY routine
   (Wait for 3 seconds to allow LED system status information to be viewed).
8. Clear all the LED output signals to save power
9. Return to the calling routine.



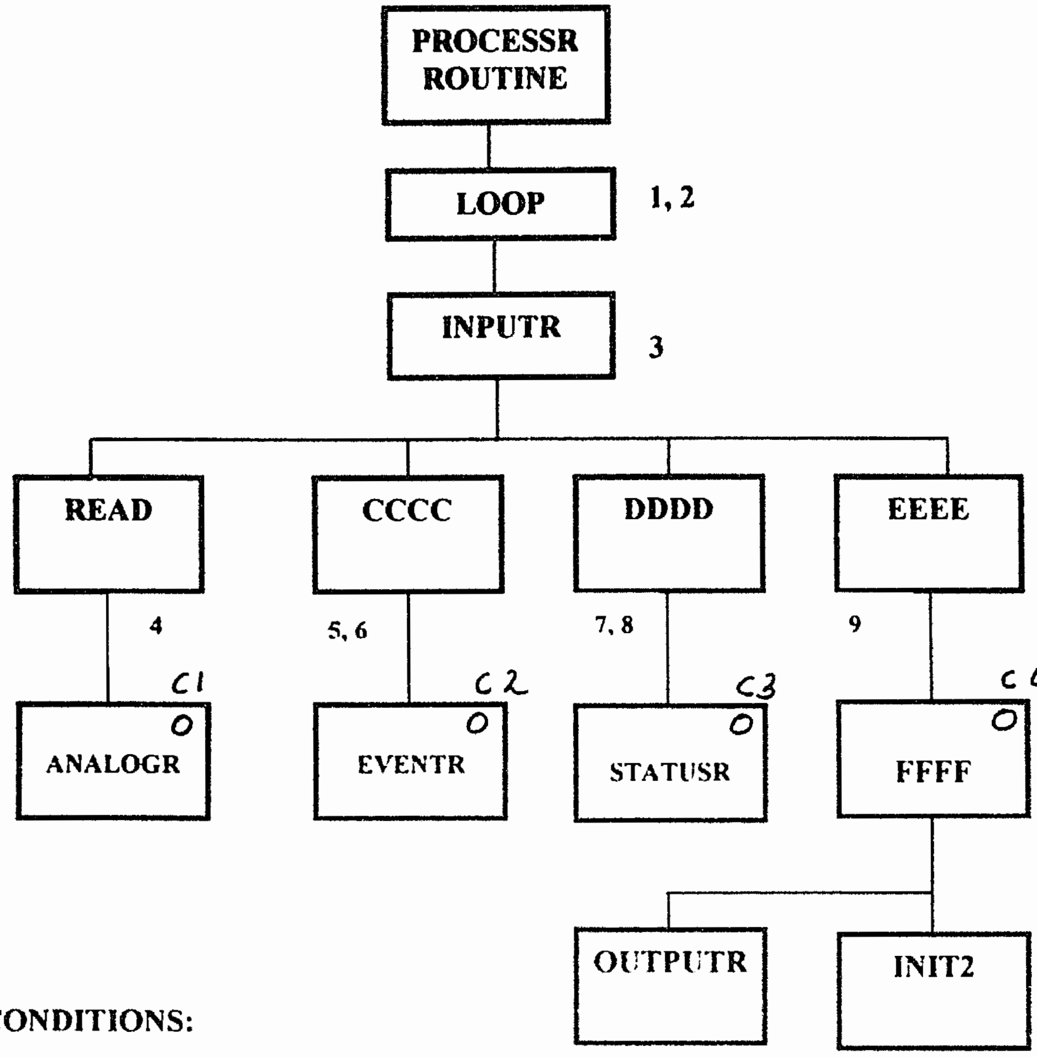
CONDITIONS:

C1. Loop around redundant code for a 3 second period.

FUNCTION:

1. Cause a 3 second delay.

THE MAIN PROCESSING



CONDITIONS:

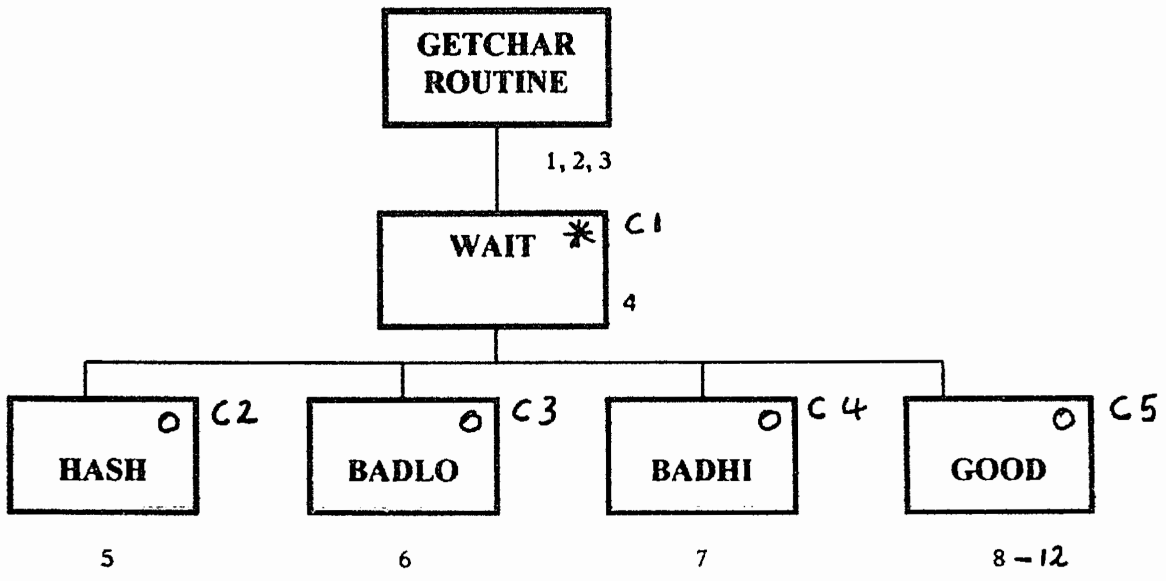
C1.	Loop aroundcode until the DNLOAD command has been detected.
C2.	Selected because the READFLAG has been set.
C3.	Selected because the EVENT flag has been set.
C4.	Selected because the DISPSTAT flag has been set.
C5.	Selected because the DNLOAD flag has been set.

FUNCTIONS:

1. Set the MONITOR status flag to indicate that the system is logging data.
2. SYNC routine (Input start of test information from the serial port).

3. INPUTR routine Check the input capture flags for push button commands.

4. Check READFLAG. If set, call the STATUSR routine.
5. Check the EVENT flag. If set, call the EVENTR routine.
6. Clear the EVENT flag.
7. Check the DISPSTAT flag. If set, call the STATUSR routine.
8. Clear the DISPSTAT flag.
9. Check the DNLOAD flag. If set, clear the MONITOR flag, call the OUTPUT routine

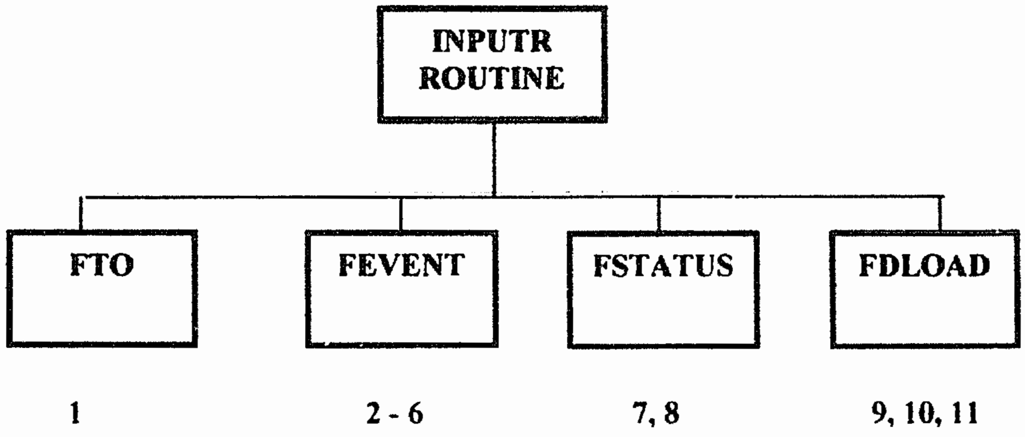


CONDITIONS:

C1.	Characters are accepted from the SCI untill an `end of block marker' is found (#).
C2.	Selected when `end of block marker' is found.
C3.	Selected when the data input from the SCI has a value less than 30h (not BCD).
C4.	Selected when the data input from the SCI has a value greater than 39h (not BCD).
C5.	Selected when BCD value are found.

FUNCTIONS:

1.	Fetch the next character from the SCI data register.
2.	If the character is an `asterisk' then reset the ONEMIN counter.
3.	Load the index register with the address of the date and time buffer.
4.	Wait for a SCI input.
5.	Clear the TRANSFER flag.
6.	Return to WAIT for another character.
7.	Return to WAIT for another character.
8.	Store character in DATEBUF.
9.	Increment the date buffer pointer.
10.	If this character does not fill the data buffer, then return to WAIT.
11.	If the data buffer was full and the character is not an '#' then, set all the status flags.
12.	Return to the SYNC routine.



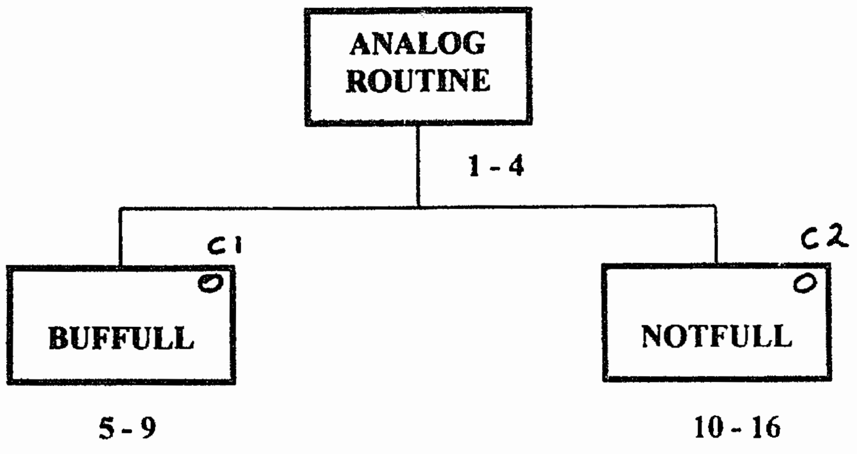
FUNCTIONS:

1. Check for a single TOF. If set, increment the TENMIN counter,
   Is TENMIN counter = ten minute value. If yes, then set the READFLAG to inform the system that the next temperature reading is due.
   Then reset the TENMIN counter register.

2. Store the cleared TENMIN count value
3. Clear the TOF flag register.
4. Test for an EVENT command
5. If an event is asked for then,wait for the push button's anti-bounce to finnish.
6. Set the EVENT flag.

7. Test for a STATUS command.
8. If a view of the system status is asked for, then set DISPSTAT flag to inform the system.

9. Test for a DNLOAD command.
10. Set the DNLOAD flag to inform the system that it has to down load its recorded data to the DTD.
11. Return to the calling program (PROCESSR).



CONDITIONS:

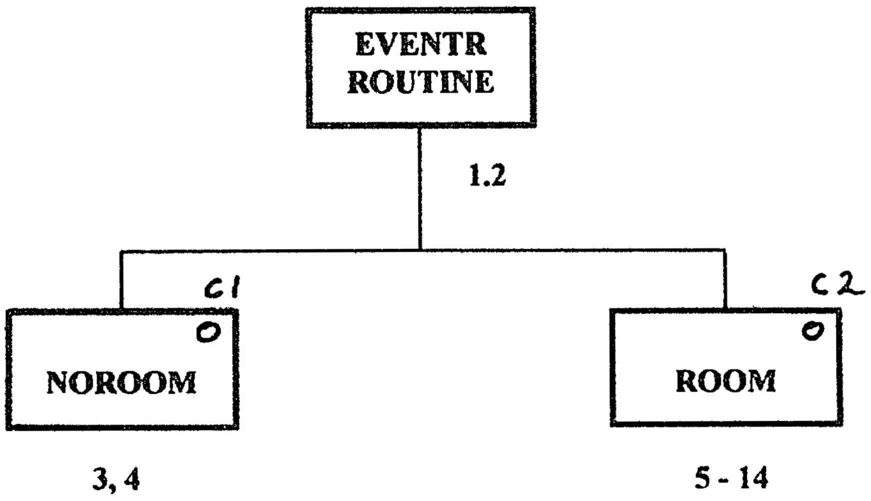
C1.	Selected if the data buffer is full.
C2.	Selected if the data buffer is not full.

FUNCTIONS:

1. Set the ADC Sub system for a multi-channel single-scan operation.
2. Load the digitised temperature reading. Store it in TEMP1.
3. Load the digitised battery reading. Store it in TEMP2.
4. Check for a BUFFER full condition.

5. Set the MEMFULL flag to inform the system that the data BUFFER is full.
6. Check the battery voltage (TEMP2) for a low condition.
7. If low, set the BATTERY status flag.
8. Clear the READFLAG status flag.
9. Return to the PROCESSR routine.

10. Store data in BUFFER and TEMP3
11. Increment BUFPTR.
12. Increment the number of temperature readings counter (AMOUNT).
13. Check the battery voltage (TEMP2) for a low condition.
14. If low, set the BATTERY status flag.
15. Clear the READFLAG status flag.
16. Return to the PROCESSR routine.



CONDITIONS:

C1.	Selected if EVENTB is full.
C2.	Selected if EVENTB is not full.

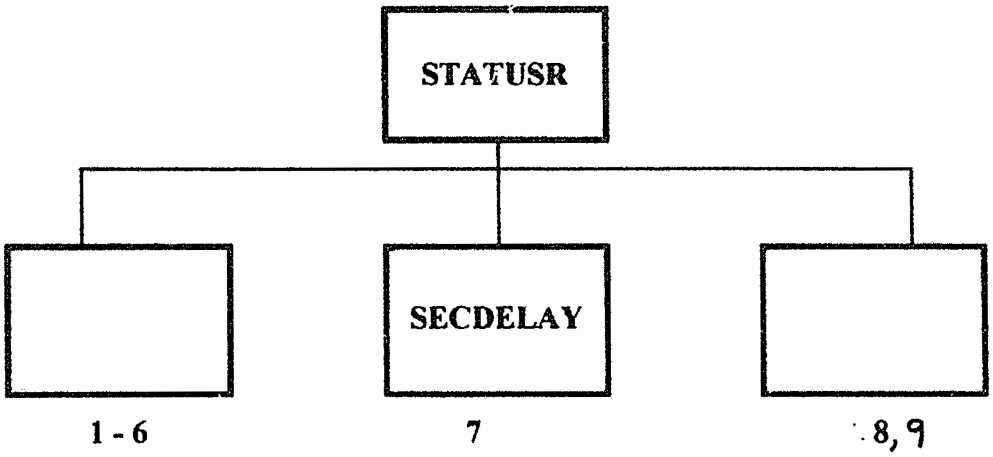
FUNCTIONS:

1. Load the index register with EVENTPTR.
2. Check EVENTPTR for its maximum value.

3. delay for the anti-bounce of the pushbutton.
4. Return to the PROCESSR routine.

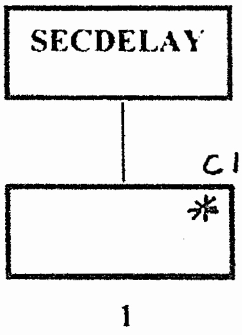
```
5.    Record the time the event happened (with respect to the start of the test
6.    Check to see if the events buffer is full.
7.        increment EVENTPTR
8.        If not full,  store the EVENTPTR value.
9.        Delay for the anti-bounce of the pushbutton.
10.       Return to the PROCESSR routine.

11.       Else (EVENTB full) set the EVENTFUL flag.
12.       Store the EVENTPTR value.
13.       Delay for the anti-bounce of the pushbutton.
14.       Return to the PROCESSR routine.
```



FUNCTIONS:

1. Clear all the LED output signals.
2. Test the MONITOR status flag.
   IF set, the appropriate LED output bit of the signal has to be set.
3. Test the MEMFULL status flag.
   IF set, the appropriate LED output bit of the signal has to be set.
4. Test the BATTERY condition status flag.
   IF set, the appropriate LED output bit of the signal has to be set.
5. Test the TRANSFER of serial data status flag.
6. Output the system status to PORTA.
7. SECDELAY routine
   (Wait for 3 seconds to allow LED system status information to be viewed).
8. Clear all the LED output signals to save power.
9. Return to the calling routine.



CONDITIONS:

C1. Loop around redundant code for a 3 second period.

FUNCTION:

1. Cause a 3 second delay.

SYNC ROUTINE

FUNCTIONS:

1. Set the TRANSFER status flag.

2. Reset the ONEMIN timeout counter

3. Clear the TOF flag register

4. Enable the SCI receiver circuit

5. Check for a serial input

6. If the SCI status register is set call GETCHAR routine

7. If the SCI status register is not set Check TOF flag

8. If TOF flag is set, (i) Clear TOF flag from the TFlg2 register,

9. (ii) Increment the ONEMIN counter,

10. (ii) Is ONEMIN = timeout value?

11. If a ONEMIN timeout has ocurred then, set all the status flags to warn the system
that a bad serial transfer has taken place

12. Return to the calling program.

OUTPUTR ROUTINE

1 - 11

FUNCTIONS:

1. Set the data TRANSFER flag.
2. Enable the SCI transmitter.
3. Clear the input capture flags.
4. Output the 'start of block marker' (*)
5. Output the device identification (HB).
6. Output the device identification (LB).
7. Output the date and time values
8. Output the data
9. Output the 'end of data marker' (@)
10. Output the time of events values.
11. Return to the PROCESSR routine.

A.3.

The Temperature Monitor Device program listing

```
  BTEXT
        TMD.ASM                         ( 24 - 4 - 93 )
        The Temperature Monitoring Device (TMD) control program.

  ETEXT



  ORG    $C000                  ;**** STORAGE BUFFERS ****

DATEBUF       RMB   14      ; ASCII  date, time and end character
BUFFER        RMB   16      ; DATA   buffer
EVENTB        RMB   16      ; EVENTS buffer; 16-bit elapsed times
BCDBUF        RMB    6      ; day, month, year, hour, minutes and seconds



  ORG    $0000                  ;** POINTERS, VARIABLES & FLAGS **

BUFFPTR     FDB    BUFFER         ; pointer used to store data
EVENTPTR    FDB    EVENTB         ; pointer used to store time of events

TENMIN      FDB    $0000          ; 10 minute counter value
ONEMIN      FDB    $0000          ;  1 minute counter value

TEMP1              FCB    $00     ; temporary storage for ADC purposes
TEMP2              FCB    $00
AMOUNT      FDB    $0000          ; number of data items stored

UPLOAD      FCB    $00            ; command flags
DNLOAD      FCB    $00
EVENT              FCB    $00
DISPSTAT           FCB    $00
MONITOR            FCB    $00     ; status flags
MEMFULL            FCB    $00
BATTERY            FCB    $00
TRANSFER           FCB    $00
EVENTFUL           FCB    $00
READFLAG           FCB    $00     ; flag set every 10 minutes; read data
TEMP3              FCB    $00     ; coded temperature value
TEMP4              FCB    $00     ; spare byte
```

```
  ORG    $C600              ;****  FIXED PARAMETERS  ****


IDENT      FDB   $5A5A             ; device identification


DATA       FDB   BUFFER            ; Start addresses of buffers used by
EVENTS     FDB   EVENTB            ; the TMD control program.
DATE       FDB   DATEBUF
CLEAR      FDB   TENMIN            ; start address of variables and flags

                                   ; area
BUFFMAX    FDB   BUFFER+16         ; end of buffer values
PTREMAX    FDB   EVENTB+16
DATEMAX    FDB   DATEBUF+13
VARYMAX    FDB   TENMIN+20


BEGCODE    FCB   $2A               ; start and end of block markers
ENDCODE    FCB   $23


TILIMIT    FDB   $0100             ; timeout values, number of TOFs
TIMEOUT    FDB   $040E


ADCDELAY   FCB   $7F               ; time for ADC subsystem to warm
                                   ; up

BATLOW     FCB   $E0               ; low voltage warning value(S.O.T.)


  ORG    $C700

LOOKUP       RMB   256
```

```
                    * ***  THE TIMER SUB SYSTEM REGISTERS  ***


TMSK2       EQU   $1024
TFLG2       EQU   $1025
TCNT        EQU   $100E


                    * ***  THE ADC SUB SYSTEM REGISTERS  ***


OPTION      EQU   $1039
ADCTL       EQU   $1030
ADR1        EQU   $1031
ADR2        EQU   $1032


          * ***  THE INPUT COMMAND / OUTPUT STATUS REGISTERS  ***


PACTL       EQU   $1026
TCTL1       EQU   $1020
TCTL2       EQU   $1021
TMSK1       EQU   $1022
TFLG1       EQU   $1023


                    * ***  THE SCI SUB SYSTEM REGISTERS  ***


BAUD        EQU   $102B
SCCR1       EQU   $102C
SCCR2       EQU   $102D
SCSR        EQU   $102E
SCDR        EQU   $102F


                    * ***  THE PARRALEL PORT REGISTERS  ***


PORTA       EQU   $1000
PORTB       EQU   $1004
PORTC       EQU   $1003
PORTD       EQU   $1008
PORTE       EQU   $100A

DDRD        EQU   $1009
```

```
  ORG    $C200                ; ****  START OF TMD PROGARM  ****


RESET   NOP                   ; cold starting place of program
        JSR    INIT1          ; initialise microcontroller sub systems
        NOP
        JSR    INIT2          ; initialise pointers and variables
        NOP
READY   LDAB   TFLG1          ; check for show_status command
        BITB   #$02
        BEQ    AAAA
        NOP
SHOW    JSR    STATUSR        ; show_status of system routine
        NOP
CLEARF  LDAA   #$FF           ; clear input capture flags
        STAA   TFLG1
        NOP
        BRA    READY
        NOP
AAAA    LDAB   TFLG1          ; check for upload_date/time command
        BITB   #$01
        BEQ    BBBB
        NOP
START   JSR    SECDLAY        ; push button anti-bounce solution
        NOP
FCLEAR  LDAA   #$FF           ; clear input capture flags
        STAA   TFLG1
        NOP
        JSR    PROCESSR       ; data logging routine
        NOP
RETURN  LDAA   #$FF           ; clear input capture flags
        STAA   TFLG1
        NOP
BBBB    BRA    READY
```

***** INITIALISE MICROCONTROLLER SUB SYSTEMS *****

```
INIT1   NOP
        LDAB  #$80          ; arrange PORTA for 4 i/ps, 4 o/ps
        STAB  PACTL
        NOP
        LDAB  #$55          ; initialise input capture for +ve edges
        STAB  TCTL2
        NOP
        LDAB  #$00          ; arrange PORTD for 8 i/ps
        STAB  DDRD
        NOP
        LDAB  OPTION        ; initialise the ADC sub system
        ANDB  #$BF
        ORAB  #$80
        STAB  OPTION
        NOP
        CLRA                . 100 uSec delay
DLAY    INCA
        CMPA  ADCDELAY           ; #$7F
        BNE   DLAY
        NOP
        LDAB  #$FF          ; set latch to enable RxD to PD0
        STAB  $4000
        NOP
        LDAB  #$30          ; 9600 baud rate
        STAB  BAUD
        NOP
        LDAB  #$00
        STAB  SCCR1         ; set SCI for 8 data bits
        NOP
        LDAB  SCCR2         ; disable SCI transmitter, receiver
        ANDB  #$03          ; and interrupts
        STAB  SCCR2
        NOP
        CLR   TMSK1         ; disable TIMER interrupts
        CLR   TCTL1         ; to avoid OUTPUT COMPARE action
        LDAB  #$FF
        STAB  TFLG1         ; clear INPUT CAPTURE flag bits
        NOP
        RTS
```

***** INITIALISE SYSTEM POINTERS AND VARIABLES *****

```
INIT2   NOP
        LDX   DATA          ; clear data buffer
GGGG    CLR   00,X
        INX
        CPX   BUFFMAX
        BLS   GGGG
        NOP
        LDX   EVENTS        ; clear events buffer
HHHH    CLR   00,X
        INX
        CPX   PTREMAX
        BLS   HHHH
        NOP
        LDX   DATE          ; clear date and time buffer
IIII    CLR   00,X
        INX
        CPX   DATEMAX
        BLS   IIII
        NOP
        LDX   CLEAR         . clear all the variables and flags
JJJJ    CLR   00,X
        INX
        CPX   VARYMAX
        BLS   JJJJ
        NOP
        LDX   #BUFFER       . initialise pointers to buffers
        STX   BUFFPTR
        LDX   #EVENTB
        STX   EVENTPTR
        NOP
        RTS
```

```
                    *****  SHOW STATUS OF SYSTEM ROUTINE  *****


STATUSR    NOP
        CLRA
        TST     MONITOR           ; test and set monitor status
        BPL     KKKK
        ORAA    #$10
KKKK    NOP
        TST     MEMFULL           ; test and set buffer memory status
        BPL     LLLL
        ORAA    #$20
LLLL    NOP
        TST     BATTERY           ; test and set battery condition status
        BPL     MMMM
        ORAA    #$40
MMMM            NOP
        TST     TRANSFER          ; test and set serial transfer status
        BPL     NNNN
        ORAA    #$80
NNNN    NOP
        STAA    PORTA             ; output the system status to PORTA
        NOP
        JSR     SECDLAY           ; output status for 3 seconds
        NOP
        CLRA
        STAA    PORTA             ; clear status output bits at PORTA
        NOP
        RTS


    ; ****  3 SECOND DELAY ROUTINE  ****


SECDLAY     NOP
            LDAA    #$10
OUTLOOP     LDX     #$FFFF
INLOOP      DEX
            BNE     INLOOP
            DECA
            BNE     OUTLOOP
            NOP
            RTS
```

***** DATA LOGGING ROUTINE *****

```
PROCESSR  NOP
          JSR    SYNCR          ; upload date and time routine
          NOP
          LDAA   #$FF           ; set monitoring data flag
          STAA   MONITOR
          NOP
LOOP      JSR    INPUTR         ; check for commands and 10 min timeout
          NOP
READ      LDAB   READFLAG       ; ready to read a new piece of data?
          BPL    CCCC
          NOP
          JSR    ANALOGR
          NOP
CCCC      LDAB   EVENT          ; has an event been signalled?
          BPL    DDDD
          NOP
          JSR    EVENTR
          NOP
          LDAA   #$00           ; clear event marker flag
          STAA   EVENT
          NOP
DDDD      LDAB   DISPSTAT       ; has a show staus command been issued?
          BPL    EEEE
          NOP
          JSR    STATUSR
          NOP
          LDAA   #$00           ; clear display status flag
          STAA   DISPSTAT
          NOP
EEEE      LDAB   DNLOAD         ; has a down load command been issued?
          BMI    FFFF
          NOP
          BRA    LOOP
          NOP
FFFF      LDAA   #$00           ; clear monitoring data flag
          STAA   MONITOR
          NOP
          JSR    OUTPUTR
          NOP
          JSR    INIT2
          NOP
          RTS
```

***** UPLOAD DATE, TIME AND START INFORMATION *****

```
SYNCR   NOP
INITF   LDAA   #$FF          ; set transferring date/time flag
        STAA   TRANSFER
        NOP
        LDX    #$0000        ; reset 1 minute timeout counter
        STX    ONEMIN
        LDAA   #$FF          ; clear TOF flag register
        STAA   TFLG2
        NOP
ENSCI   LDAB   #$04          ; enable SCI receiver
        STAB   SCCR2
        NOP
HHH     LDAB   SCSR          ; wait for a serial input
        ANDB   #$20
        BNE    GETCHAR
        TST    TFLG2         ; has a TOF occurred
        BPL    HHH
        LDAA   #$FF
        STAA   TFLG2         ; clear TOF from TIMER flag register
        LDY    ONEMIN        ; increment timeout counter
        INY
        STY    ONEMIN
        CPY    TIMEOUT       ; has a timeout occurred
        BLS    HHH
SETFLAGS   LDAA   #$FF       ; set all OUTPUT COMPARE flag bits
        STAA   MONITOR
        STAA   MEMFULL
        STAA   BATTERY
        STAA   TRANSFER
        NOP
        RTS

GETCHAR    NOP
        LDAB   SCDR          ; receive first character
        CMPB   #$2A          ; is it the start of block marker '*'
        BNE    HHH
        NOP
        LDY    #$0000        ; reset 1 minute timeout counter
        STY    ONEMIN
        LDX    #DATEBUF      ; address of date and time buffer
        NOP
```

```
WAIT    LDAB   SCSR
        ANDB   #$20
        BEQ    WAIT
        NOP
        LDAA   SCDR
        CMPA   #$23              ; end of block marker `#'
HASH    BEQ    BACK
        CMPA   #$30              ; ASCII BCD   .racter `0'
BADLO          BLT    WAIT
        CMPA   #$39              ; ASC11 BCD character `1'
BADHI   BHI    WAIT
        NOP
GOOD    STAA   00,X
        INX
        CPX    DATEMAX
        BLT    WAIT
        NOP
        JMP    SETFLAGS          ; no end of block marker
BACK    NOP
        LDAA   #$00              clear transferring date/time flag
        STAA   TRANSFER
        NOP
        RTS
```

```
***** CHECK P.B. COMMANDS AND 10 MINUTE TIMEOUT *****

INPUTR NOP
FTO    TST   TFLG2          ; check for a timer overflow (TOF)
       BPL   NOTIME
       LDX   TENMIN
       INX
       CPX   TILIMIT
       BLT   FEVENT
       LDAA  #$FF           ; set time to read a temperature flag
       STAA  READFLAG
       LDX   #$0000         ; reset 10 minute counter
FEVENT STX   TENMIN
       LDAB  #$80           ; clear TOF flag
       STAB  TFLG2
NOTIME NOP
       LDAB  TFLG1          ; test for an event command
       BITB  #$04
       BEQ   FSTATUS
       NOP
       JSR   SECDLAY        ; push button anti-bounce solution
       NOP
       LDAA  #$FF           ; set the appropriate flag
       STAA  EVENT
       STAA  TFLG1
FSTATUS NOP
       LDAB  TFLG1          ; test for a status command
       BITB  #$02
       BEQ   FDLOAD
       LDAA  #$FF           ; set the appropriate flag
       STAA  DISPSTAT
       STAA  TFLG1
FDLOAD NOP
       LDAB  TFLG1          ; test for an down load command
       BITB  #$01
       BEQ   WWWW
       LDAA  #$FF           ; set the appropriate flag
       STAA  DNLOAD
       STAA  TFLG1
WWWW   NOP
       RTS
```

***** READ NEXT PIECE OF DATA FROM THE ADC *****

```
ANALOGR   NOP
          LDAA   #$10           ; select multi-channel single scan
          STAA   ADCTL          ; ADR1 - ADR4
          NOP
OOOO      TST    ADCTL
          BPL    OOOO
          LDAA   ADR1           ; digitised temperature reading
          LDAB   ADR2           ; digitised battery voltage
          STAB   TEMP2          ; current battery voltage reading
          LDY    #LOOKUP
          STAA   TEMP1          ; current temperature reading

          BTEXT

PPPP      BEQ    QQQQ
          INY                   ; IX register is used as a pointer
          DECA                  ; into the LOOKUP table
          BRA    PPPP
          NOP
QQQQ      LDAA   00,Y           ; the appropriate coded Celsius value

          ETEXT

          LDX    BUFFPTR
          CPX    BUFFMAX        ; is buffer full
          BNE    NOTFULL
BUFFULL   LDAA   #$FF           ; set buffer full flag (MEMFULL)
          STAA   MEMFULL
          BRA    MISSOUT
NOTFULL   STAA   00,X           ; store coded value in data buffer
          STAA   TEMP3          ; store coded value in TEMP3
          INX
          STX    BUFFPTR        ; pointer to next buffer location
          NOP
          LDX    AMOUNT         ; number of temperature readings counter
          INX
          STX    AMOUNT
          NOP
MISSOUT   CMPB   BATLOW         ; is battery voltage alright
          BHI    BATOK
          LDAA   #$FF           ; set battery status low flag
          STAA   BATTERY
          NOP
BATOK     CLR    READFLAG       ; job done
          NOP
          RTS
```

***** RECORD TIME OF EVENT *****

```
EVENTR        NOP
        LDX    EVENTPTR        ; pointer used to record event times
        CPX    PTREMAX
        BHS    NOROOM          ; is events buffer full
        NOP
ROOM    LDAA   AMOUNT          ; high byte
        LDAB   AMOUNT+1        ; low  byte
        STAA   00,X
        INX
        STAB   00,X
        INX
        CPX    PTREMAX         ; is events buffer full
        BNE    SSSS
        NOP
        LDAA   #$FF            ; set events buffer full flag (EVENTFUL)
        STAA   EVENTFUL
SSSS    NOP
        STX    EVENTPTR
NOROOM        NOP
        JSR    SECDLAY         . push button anti-bounce solution
        NOP
        RTS
```

***** DOWN LOAD DATA TO SERIAL PORT *****

```
OUTPUTR   NOP
          LDAA   #$FF          ; set transferring data flag
          STAA   TRANSFER
          NOP
          LDAB   #$08          ; enable transmitter
          STAB   SCCR2
          NOP
          LDAA   #$FF          ; clear INPUT CAPTURE flag register
          STAA   TFLG1
          NOP
AAA       LDAB   SCSR          ; ready to output
          ANDB   #$80
          BEQ    AAA
          LDAA   #$2A          ; output start of block marker
          STAA   SCDR
          NOP
BBB       LDAB   SCSR          ; ready to output
          ANDB   #$80
          BEQ    BBB
          LDAA   IDENT         ; output device identification; high byte
          STAA   SCDR
          NOP
CCC       LDAB   SCSR          ; ready to output
          ANDB   #$80
          BEQ    CCC
          LDAA   IDENT+1       ; output device identification; high byte
          STAA   SCDR
          NOP
          LDX    #DATEBUF      ; output date and time
DDD       LDAB   SCSR          ; ready to output
          ANDB   #$80
          BEQ    DDD
          LDAA   00,X          ; next byte
          STAA   SCDR
          INX
          CPX    DATEMAX
          BLS    DDD           ; last byte?
          NOP
          LDX    #BUFFER       ; output data
```

```
EEE     LDAB   SCSR             ; ready to output
        ANDB   #$80
        BEQ    EEE
        LDAA   00,X             ; next byte
        STAA   SCDR
        INX
        CPX    BUFFMAX
        BLS    EEE              ; last byte?
        NOP
        LDX    #EVENTB          ; output events buffer contents
FFF     LDAB   SCSR             ; ready to output
        ANDB   #$80
        BEQ    FFF
        LDAA   00,X             ; next byte
        STAA   SCDR
        INX
        CPX    PTREMAX
        BLS    FFF         ; last byte?
        NOP
GGG     LDAB   SCSR
        ANDB   #$80
        BEQ    GGG
        LDAA   #$23             ; end of block marker
        STAA   SCDR
        NOP
        LDAA   #$00             ; clear transferring data flag
        STAA   TRANSFER
        NOP
        RTS


  END
```

APPENDIX B

The Data Transfer Device

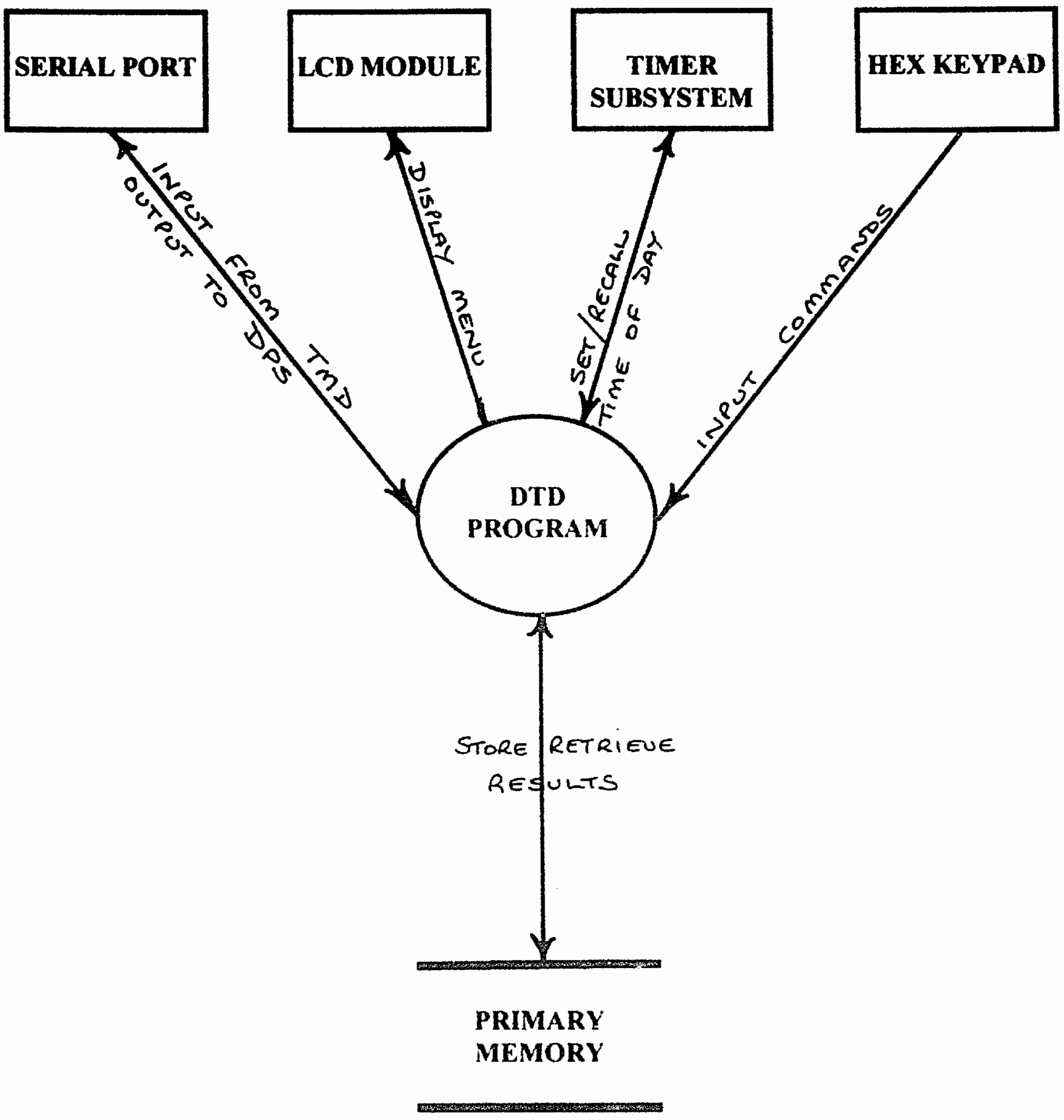
This section contains data flow diagrams, Jackson structure diagrams and program listing for the data transfer device.

B.1.

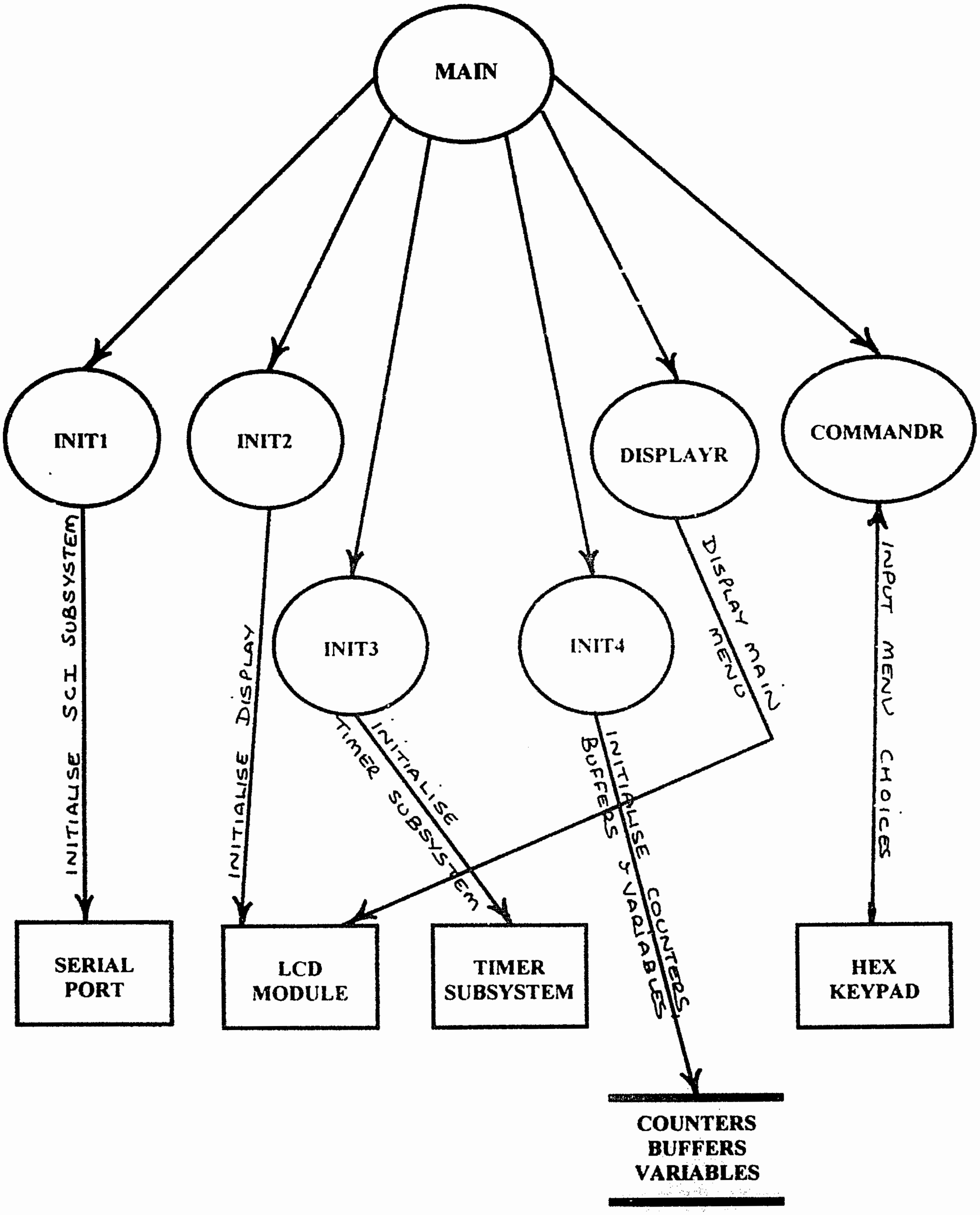
The Data Transfer Device

data flow diagrams

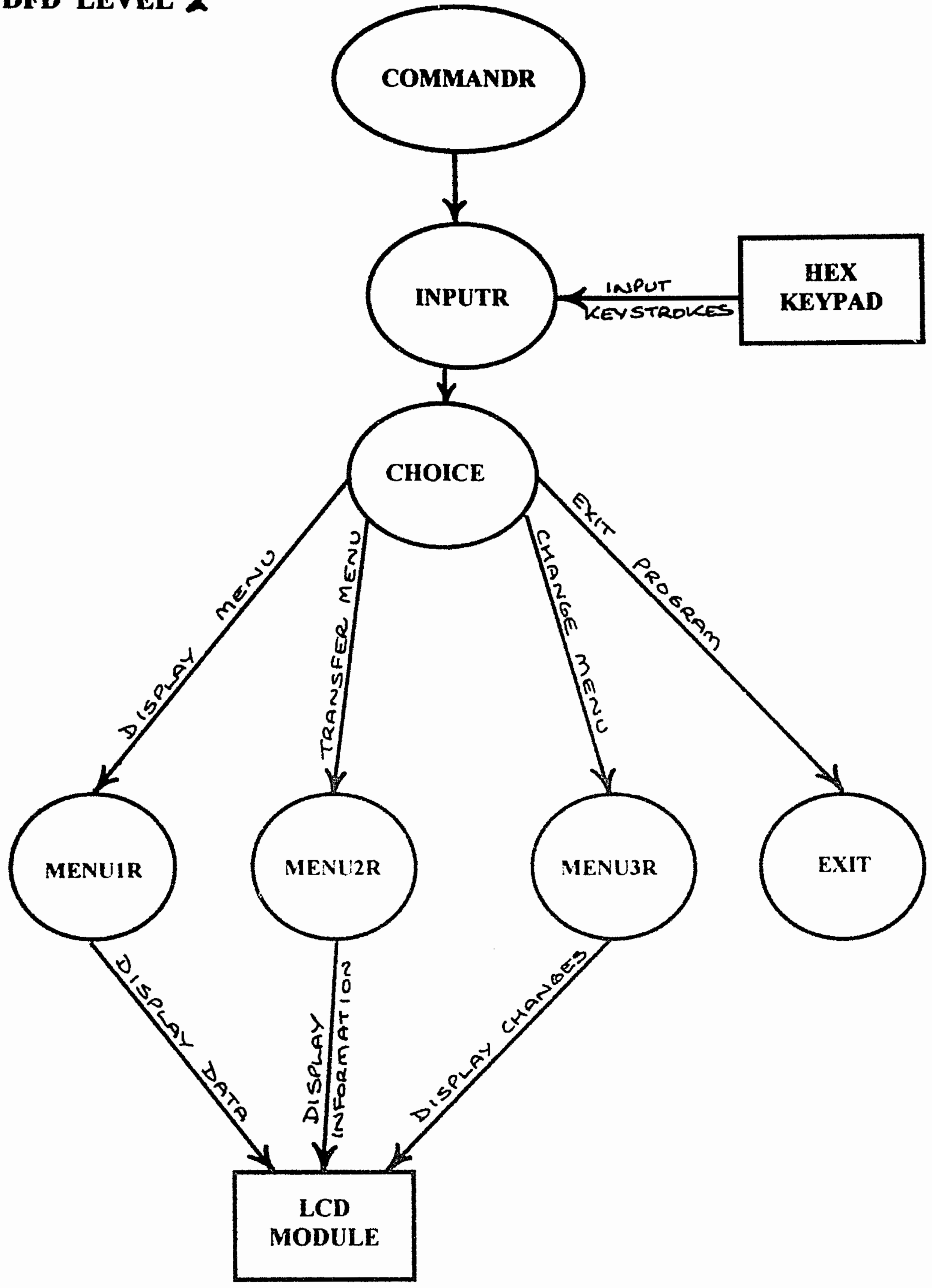
DFD LEVEL 0


SERIAL PORT
LCD MODULE
TIMER SUBSYSTEM
HEX KEYPAD
INPUT FROM TMD
OUTPUT TO DPS
DISPLAY MENU
SET/RECALL TIME OF DAY
INPUT COMMANDS
DTD PROGRAM
STORE RETRIEVE RESULTS
PRIMARY MEMORY

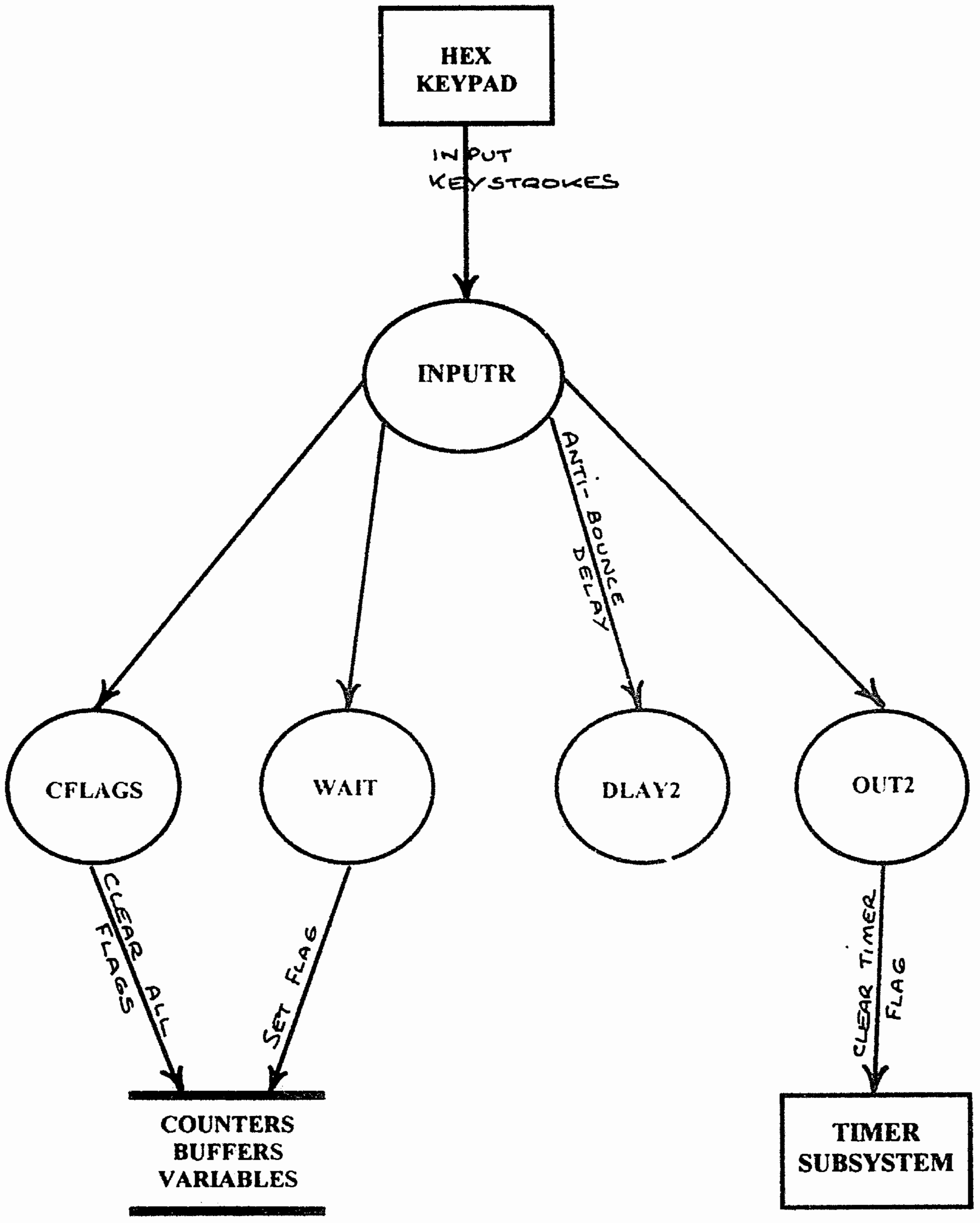
DFD LEVEL 1


MAIN
INIT1
INIT2
INIT3
INIT4
DISPLAYR
COMMANDR
INITIALISE SCI SUBSYSTEM
INITIALISE DISPLAY
TIMER SUBSYSTEM
INITIALISE TIMER SUBSYSTEM
INITIALISE COUNTERS BUFFERS & VARIABLES
DISPLAY MAIN MENU
INPUT MENU CHOICES
SERIAL PORT
LCD MODULE
TIMER SUBSYSTEM
HEX KEYPAD
COUNTERS BUFFERS VARIABLES

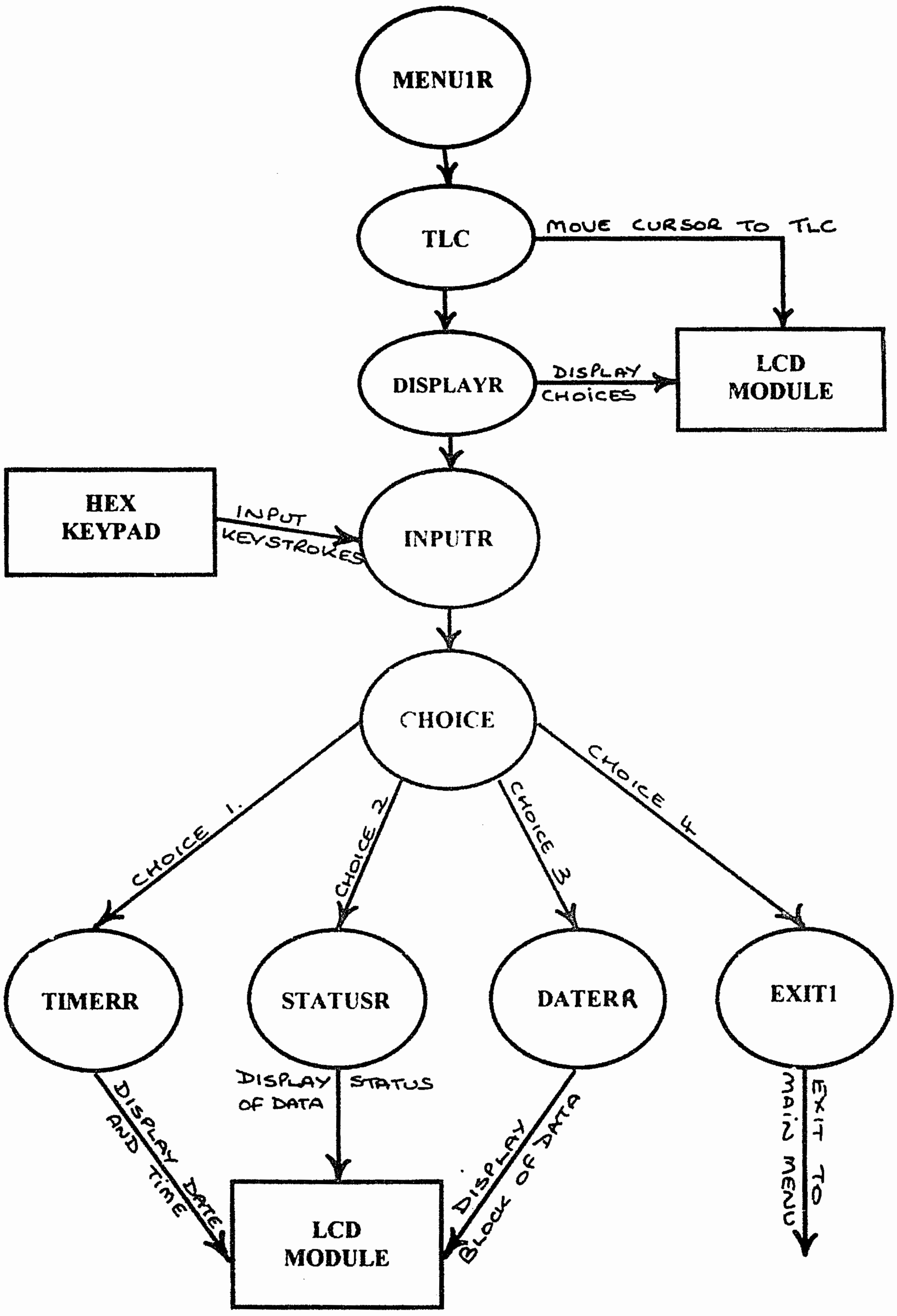
DFD LEVEL 2


COMMANDR
INPUTR
INPUT KEYSTROKES
HEX KEYPAD
CHOICE
DISPLAY MENU
TRANSFER MENU
CHANGE MENU
EXIT PROGRAM
MENU1R
MENU2R
MENU3R
EXIT
DISPLAY DATA
DISPLAY INFORMATION
DISPLAY CHANGES
LCD MODULE

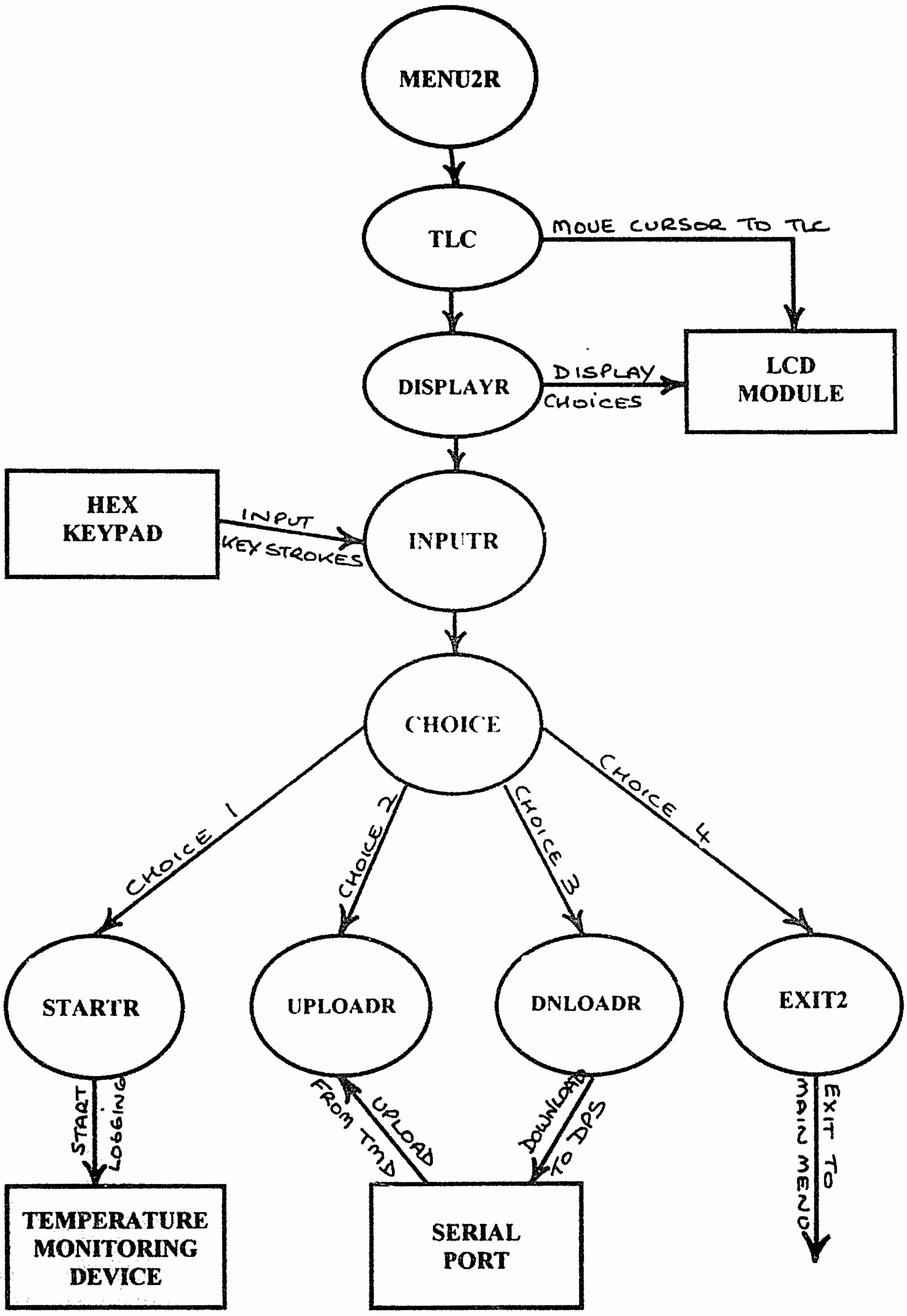
DFD LEVEL 3


HEX KEYPAD
INPUT KEYSTROKES
INPUTR
ANTI-BOUNCE DELAY
CFLAGS
WAIT
DLAY2
OUT2
CLEAR ALL FLAGS
SET FLAG
CLEAR TIMER FLAG
COUNTERS BUFFERS VARIABLES
TIMER SUBSYSTEM

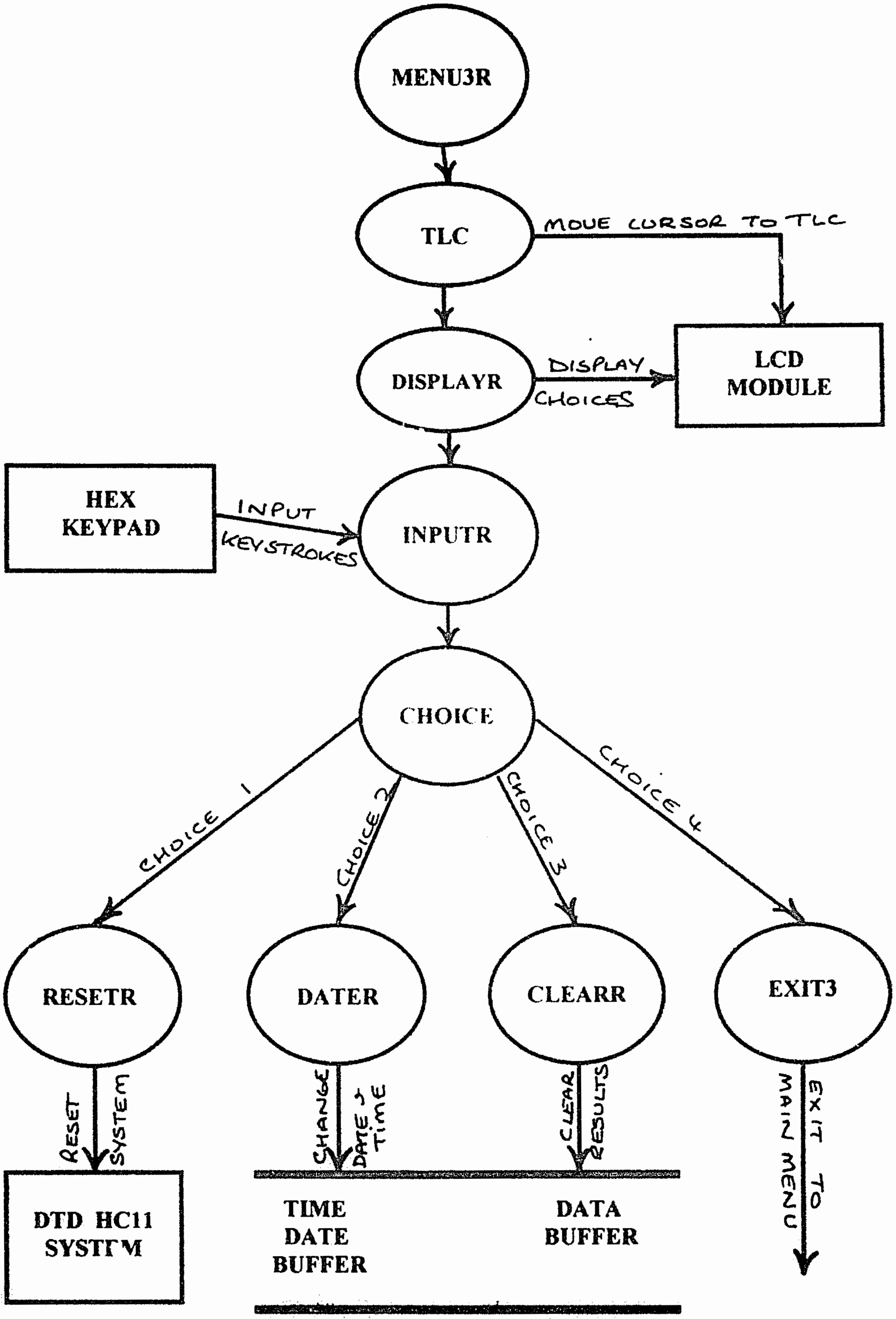
DFD LEVEL 3


MENU1R
TLC
MOVE CURSOR TO TLC
DISPLAYR
DISPLAY CHOICES
LCD MODULE
HEX KEYPAD
INPUT KEYSTROKES
INPUTR
CHOICE
CHOICE 1.
CHOICE 2
CHOICE 3
CHOICE 4
TIMERR
STATUSR
DATERR
EXIT1
DISPLAY DATE AND TIME
DISPLAY STATUS OF DATA
DISPLAY BLOCK OF DATA
EXIT TO MAIN MENU
LCD MODULE

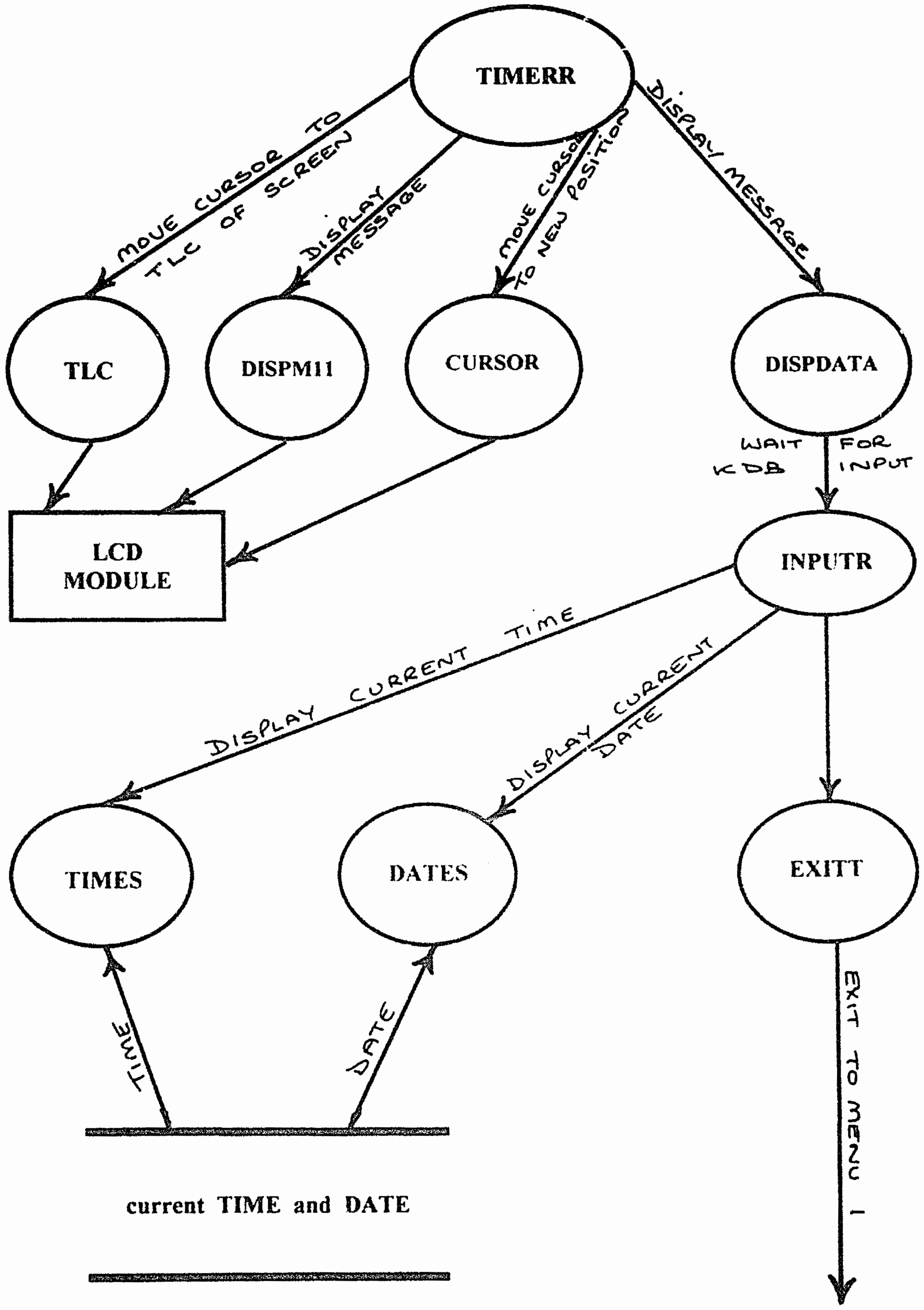
DFD LEVEL 3


MENU2R
TLC
MOVE CURSOR TO TLC
DISPLAYR
DISPLAY CHOICES
LCD MODULE
HEX KEYPAD
INPUT KEY STROKES
INPUTR
CHOICE
CHOICE 1
CHOICE 2
CHOICE 3
CHOICE 4
STARTR
UPLOADR
DNLOADR
EXIT2
START LOGGING
TEMPERATURE MONITORING DEVICE
FROM TMD
UPLOAD
DOWNLOAD TO DPS
SERIAL PORT
EXIT TO MAIN MENU

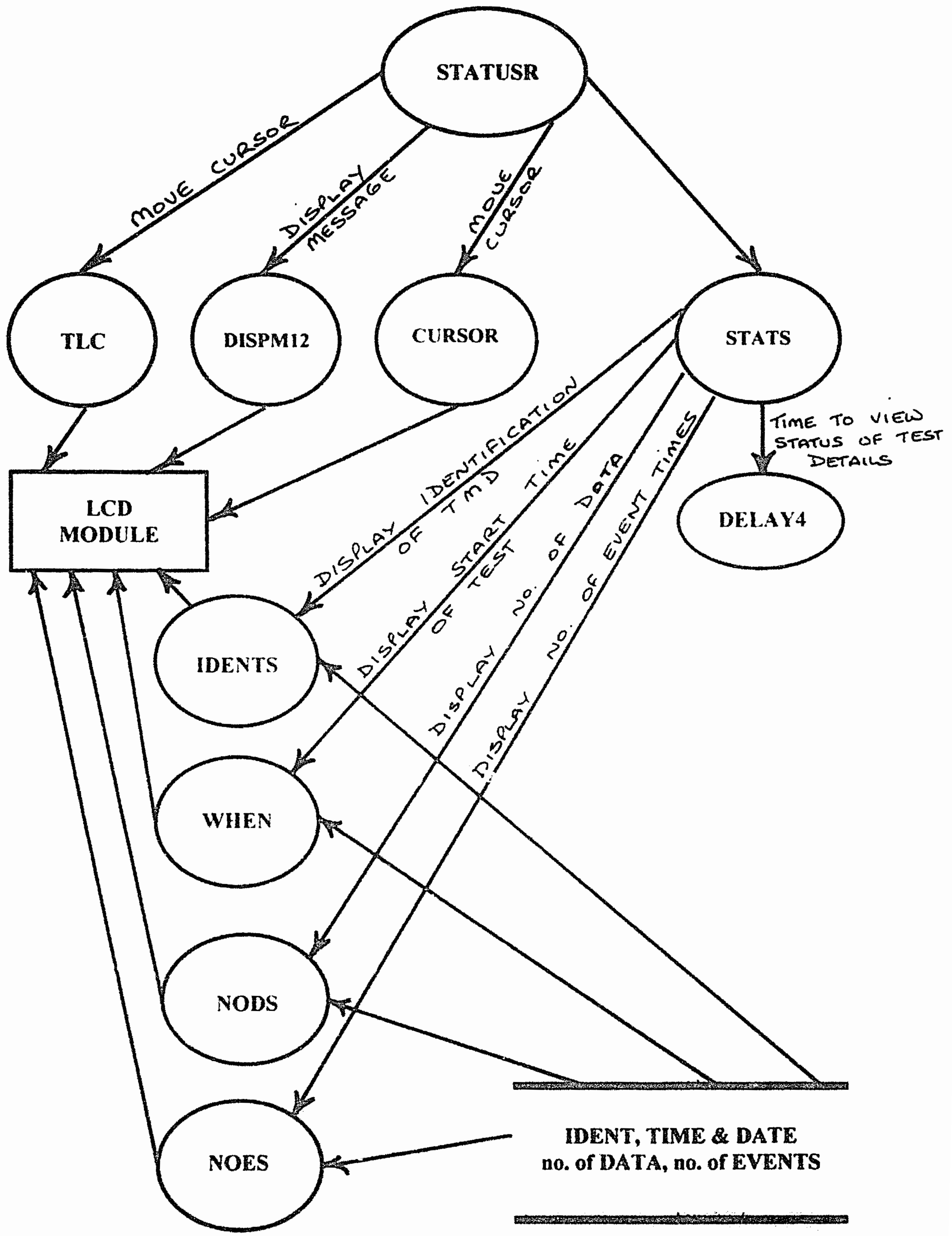
DFD LEVEL 3


MENU3R
TLC
MOVE CURSOR TO TLC
DISPLAYR
DISPLAY CHOICES
LCD MODULE
HEX KEYPAD
INPUT KEYSTROKES
INPUTR
CHOICE
CHOICE 1
CHOICE 2
CHOICE 3
CHOICE 4
RESETR
DATER
CLEARR
EXIT3
RESET SYSTEM
CHANGE DATE + TIME
CLEAR RESULTS
EXIT TO MAIN MENU
DTD HC11 SYSTEM
TIME DATE BUFFER
DATA BUFFER

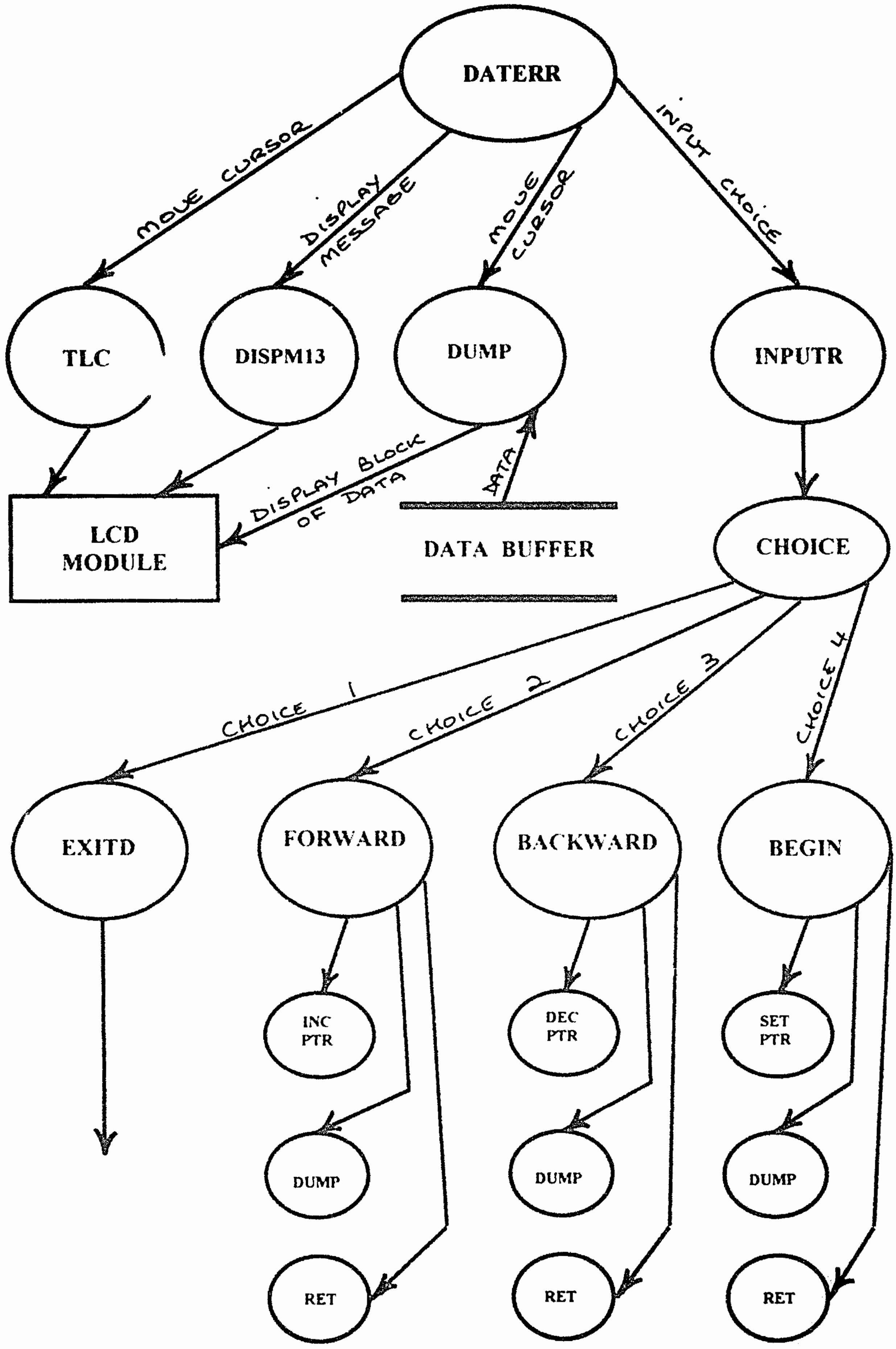
DFD LEVEL 4


TIMERR
MOVE CURSOR TO TLC OF SCREEN
DISPLAY MESSAGE
MOVE CURSOR TO NEW POSITION
DISPLAY MESSAGE
TLC
DISPM11
CURSOR
DISPDATA
WAIT KDB FOR INPUT
LCD MODULE
INPUTR
DISPLAY CURRENT TIME
DISPLAY CURRENT DATE
TIMES
DATES
EXITT
TIME
DATE
EXIT TO MENU 1
current TIME and DATE

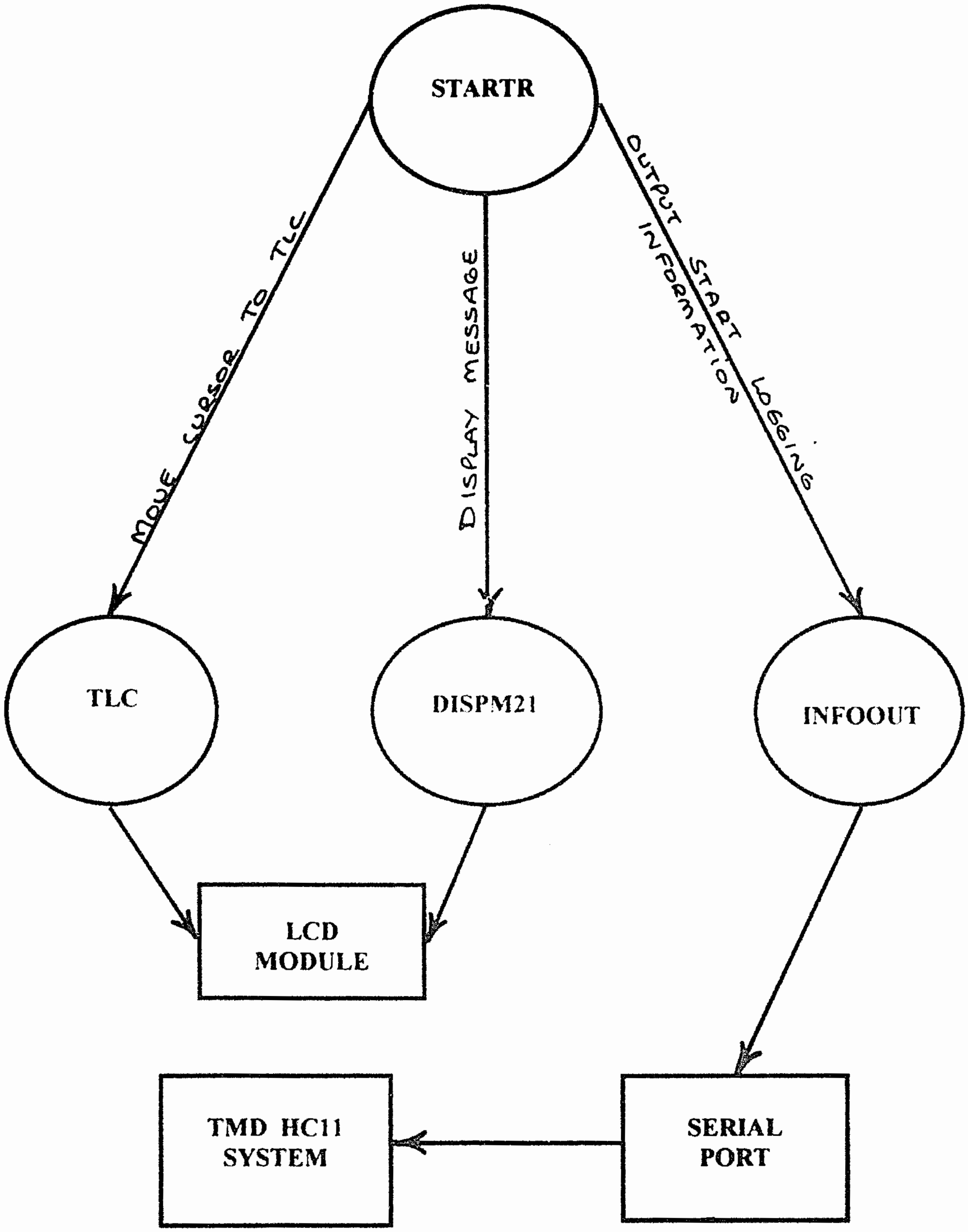
DFD LEVEL 4


STATUSR
MOVE CURSOR
DISPLAY MESSAGE
MOVE CURSOR
TLC
DISPM12
CURSOR
STATS
TIME TO VIEW STATUS OF TEST DETAILS
DELAY4
LCD MODULE
DISPLAY IDENTIFICATION OF TMD
DISPLAY START TIME OF TEST
DISPLAY NO. OF DATA
DISPLAY NO. OF EVENT TIMES
IDENTS
WHEN
NODS
NOES
IDENT, TIME & DATE
no. of DATA, no. of EVENTS

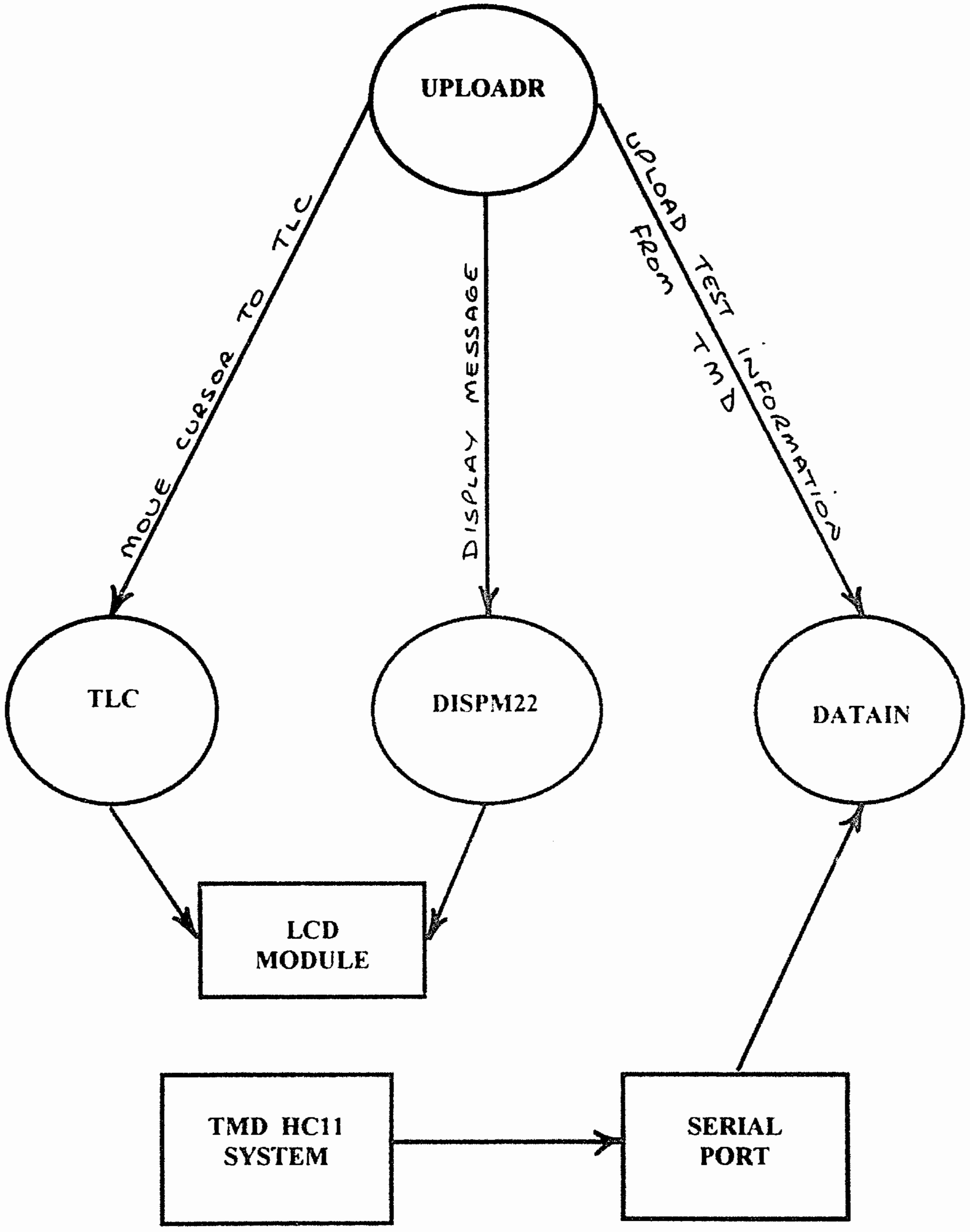
DFD LEVEL 4


DATERR
MOVE CURSOR
DISPLAY MESSAGE
MOVE CURSOR
INPUT CHOICE
TLC
DISPM13
DUMP
INPUTR
LCD MODULE
DISPLAY BLOCK OF DATA
DATA
DATA BUFFER
CHOICE
CHOICE 1
CHOICE 2
CHOICE 3
CHOICE 4
EXITD
FORWARD
BACKWARD
BEGIN
INC PTR
DEC PTR
SET PTR
DUMP
DUMP
DUMP
RET
RET
RET

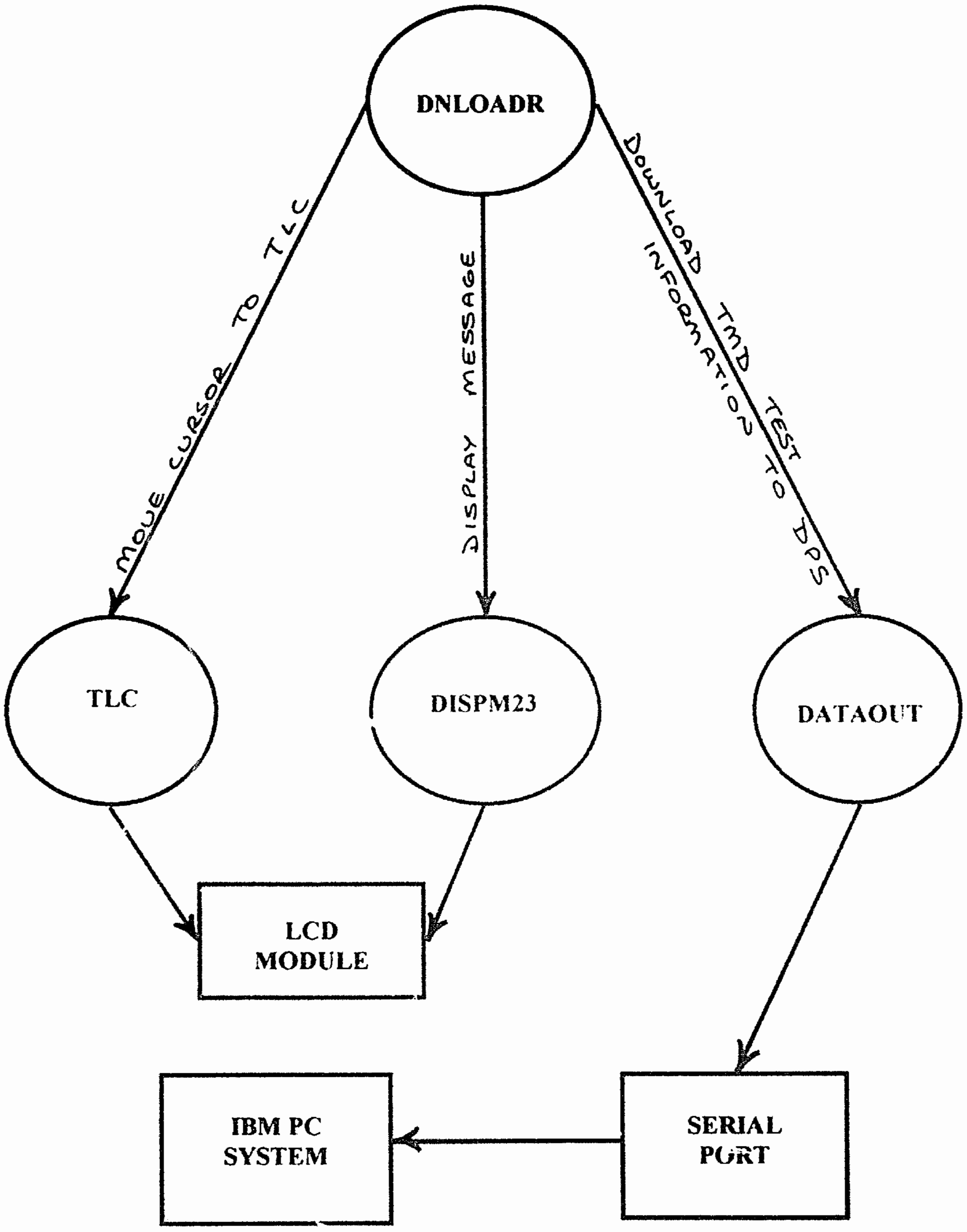
DFD LEVEL 4


STARTR
MOVE CURSOR TO TLC
DISPLAY MESSAGE
OUTPUT START LOGGING INFORMATION
TLC
DISPM21
INFOOUT
LCD MODULE
TMD HC11 SYSTEM
SERIAL PORT

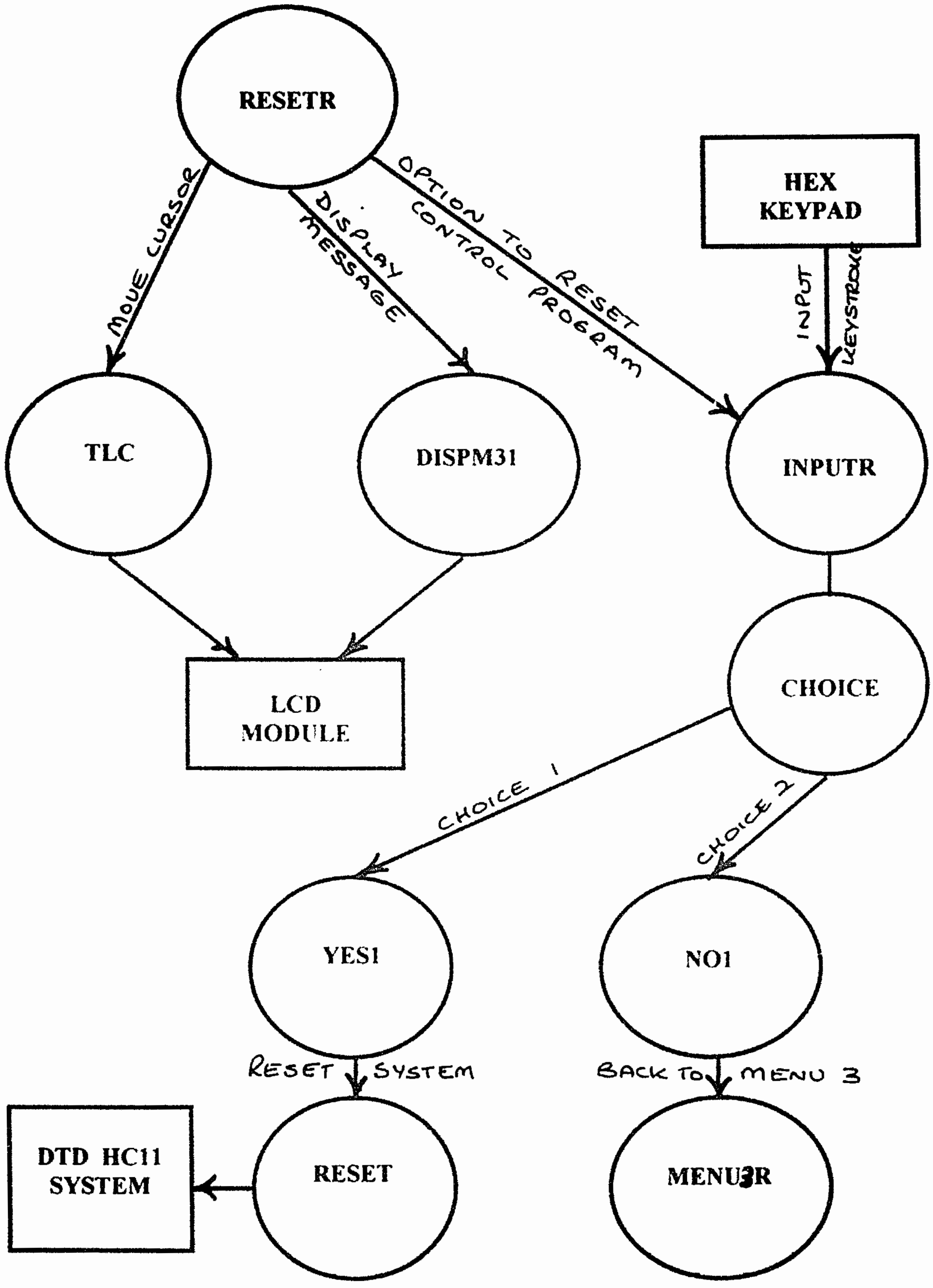
DFD LEVEL 4


UPLOADR
MOVE CURSOR TO TLC
DISPLAY MESSAGE
UPLOAD TEST INFORMATION FROM TMD
TLC
DISPM22
DATAIN
LCD MODULE
TMD HC11 SYSTEM
SERIAL PORT

DFD LEVEL 4

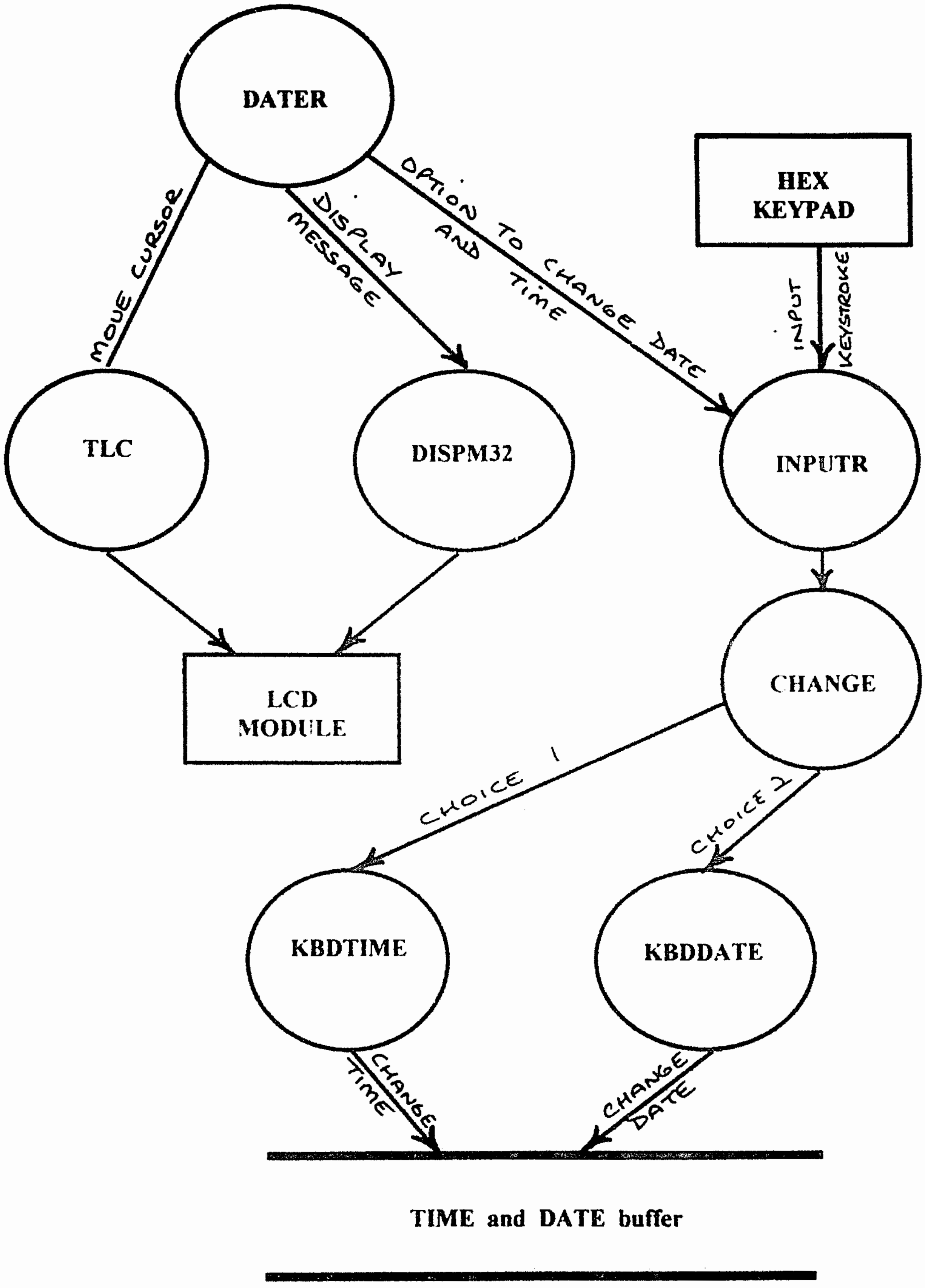

DNLOADR
MOVE CURSOR TO TLC
DISPLAY MESSAGE
DOWNLOAD TMD TEST INFORMATION TO DPS
TLC
DISPM23
DATAOUT
LCD MODULE
IBM PC SYSTEM
SERIAL PORT

DFD LEVEL 4

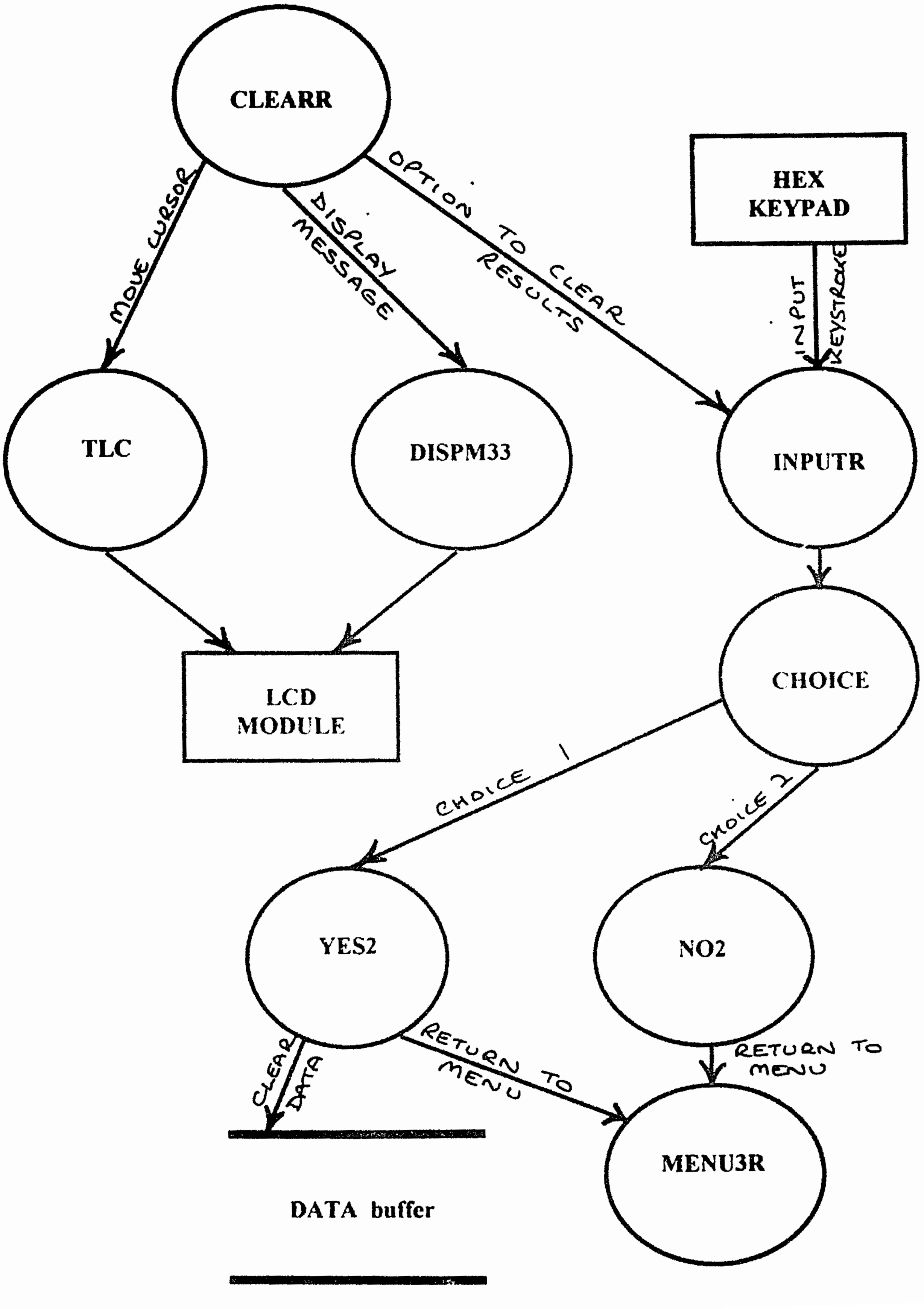


RESETR
MOVE CURSOR
DISPLAY MESSAGE
OPTION TO RESET CONTROL PROGRAM
HEX KEYPAD
INPUT
KEYSTROKE
TLC
DISPM31
INPUTR
LCD MODULE
CHOICE
CHOICE 1
CHOICE 2
YES1
NO1
RESET SYSTEM
BACK TO MENU 3
DTD HC11 SYSTEM
RESET
MENU3R

DFD LEVEL 4


DATER
HEX KEYPAD
MOVE CURSOR
DISPLAY MESSAGE
OPTION TO CHANGE DATE AND TIME
INPUT KEYSTROKE
TLC
DISPM32
INPUTR
LCD MODULE
CHANGE
CHOICE 1
CHOICE 2
KBDTIME
KBDDATE
CHANGE TIME
CHANGE DATE
TIME and DATE buffer

DFD LEVEL 4

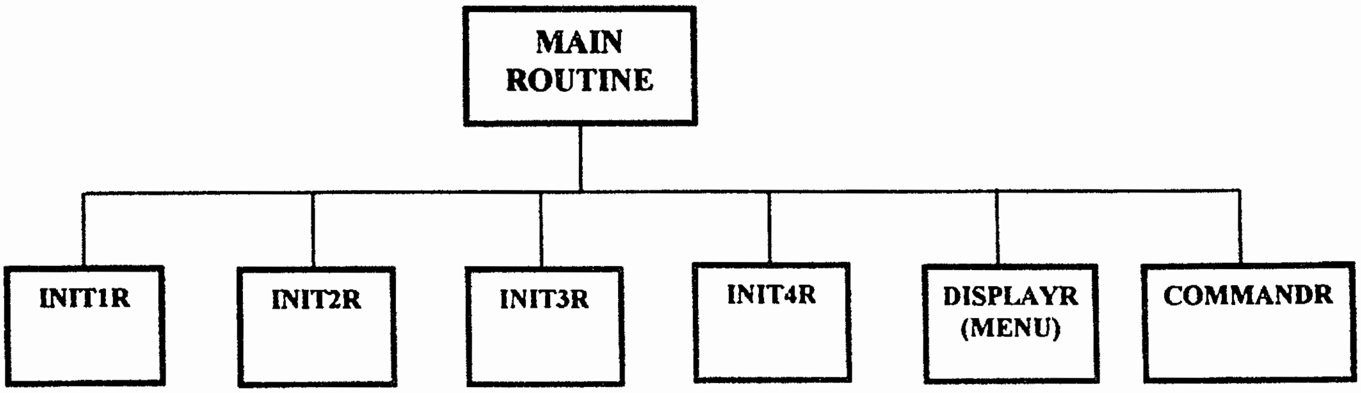

CLEARR
MOVE CURSOR
DISPLAY MESSAGE
OPTION TO CLEAR RESULTS
HEX KEYPAD
INPUT KEYSTROKE
TLC
DISPM33
INPUTR
LCD MODULE
CHOICE
CHOICE 1
CHOICE 2
YES2
NO2
CLEAR DATA
RETURN TO MENU
RETURN TO MENU
MENU3R
DATA buffer

B.2.

The Data Transfer Device

Jackson structure diagrams

DTD PROGRAM STRUCTURE



1, 2, 3, 4

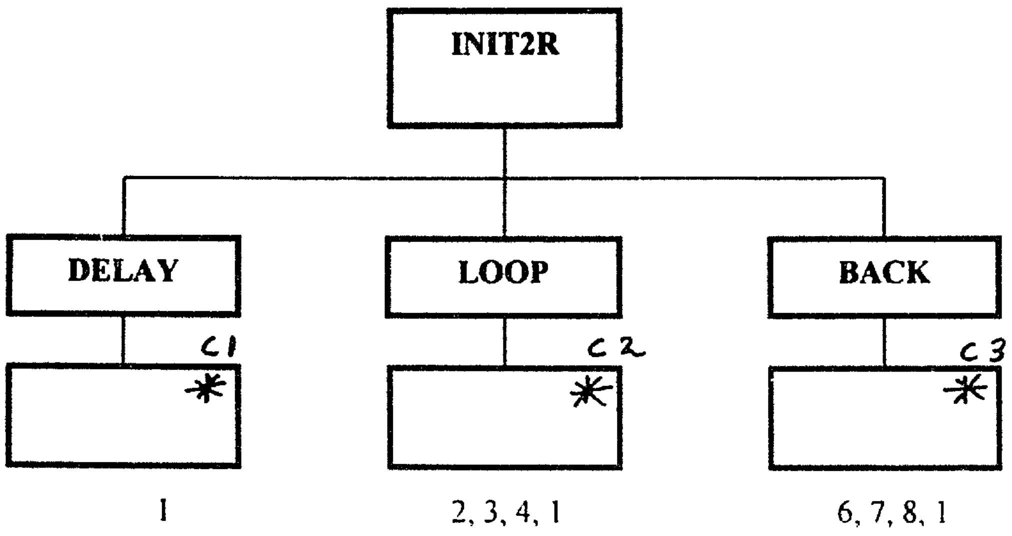
FUNCTIONS:

1. Enable receive data signal (RxD) to be latched to pin PD0.
2. Set baud rate of SCI subsystem to 9600.
3. Set SCI subsystem for 8 data bits.
4. Disable SCI transmitter and SCI receiver.

1, 2

FUNCTIONS:

1. Clear all input capture flags.
2. Program input capture interface to detect positive edges.



FUNCTIONS:

1. Loop around redundant code 36 times to cause a 40 uSec. delay.

2. Fetch next 8-bit command
3. Fetch time value for delay routine.
4. OUTPUT subroutine (Output an 8-bit LCD command)
5. Delay to allow LCD module to process the command.

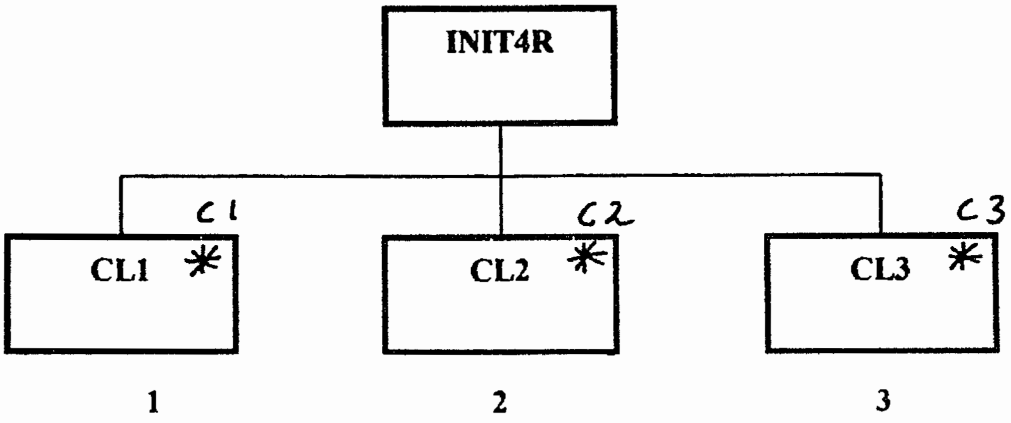
6. Fetch next 4-bit command.
7. Fetch time value for delay routine.
8. OUTPUT2 subroutine (Output a 4-bit LCD command).
9. Delay to allow LCD module to process the command.

CONDITIONS:

C1. Loop untill parameter passed to DELAY routine, to cause a 16 mSec delay, is decremented to zero.

C2. Repeat untill four 8-bit commands sent to LCD module.

C3. Repeat untill four 4-bit commands sent to LCD module.

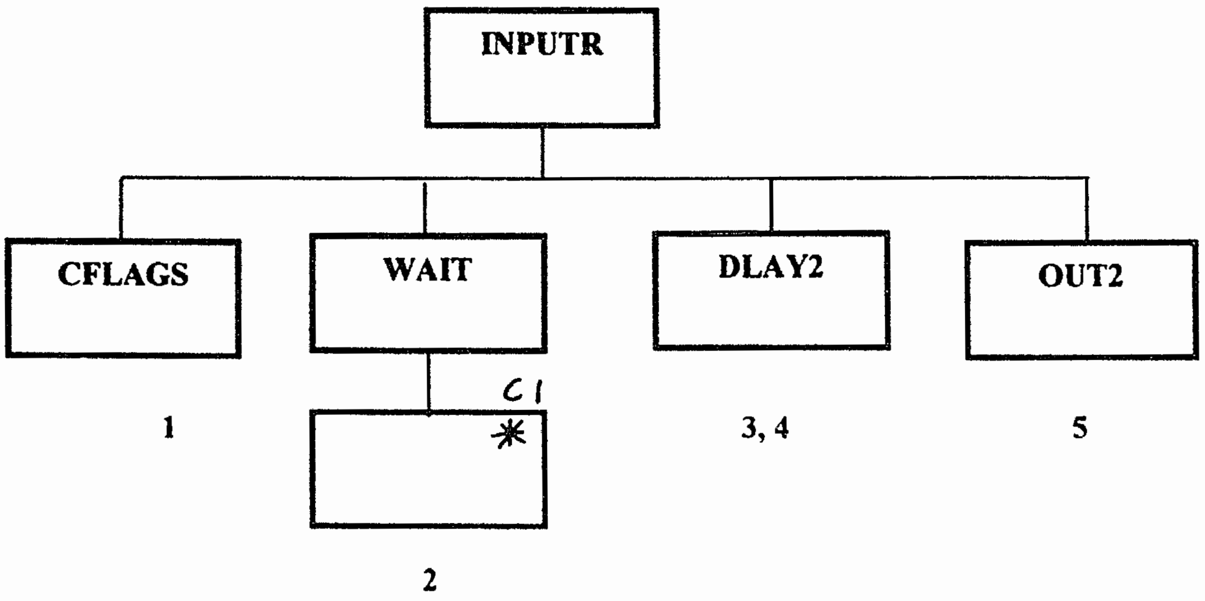


FUNCTIONS:

1.	Clear counter values.
2.	Store space characters in HEXBUF.
3.	Clear date, data and events buffers

CONDITIONS:

C1.	Locations cleared untill (HEXBUF - DATA) cleared.
C2	Store space characters untill 4 locations filled.
C3.	Locations cleared untill (TIME - DATEBUF) cleared.



FUNCTIONS:

1.	Clear input capture flags.
2.	receive an input caputre signal
3.	Read KBD input value.
4.	Store KBD input value at DATA
5.	Clear input capture flag.

CONDITIONS:

C1.	Loop untill the input capture flag (IC1) has been set; by a key press.

CONVERT

1, 2, 3, 4, 5

FUNCTIONS:

1. Fetch KBD character from DATA.
2. Distinguish between characters 0 - 9 and A - F.
3. Add the value 7 to the KBD character if A - F.
4. Convert KBD character to an ASCII character.
5. Store ASCII character in HEXBUF.

TLC

6, 7, 8

FUNCTIONS:

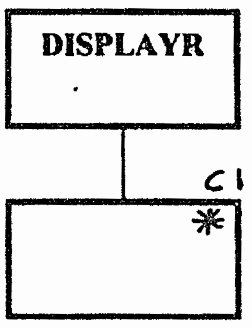
6. Load the screen address for the top left hand corner.
7. OUTPUT2 subroutine (Output a new screen address command).
8. Delay to allow LCD module to process command

CURSOR

9, 10, 11

FUNCTIONS:

9. Parameter for new screen address passed via accA, and stored in TEMP1.
10. OUTPUT2 subroutine (Output a new cursor position command).
11. Delay to allow LCD module to process command.



1, 2, 3

FUNCTIONS:

1. Fetch the next ASCII character
2. SCREEN subroutine (Send character to LCD screen).
3. Delay (40 uSec.) for LCD module to process character

CONDITIONS:

C1 Loop untill last character ($) detected

4, 5, 6, 7, 8, 9

FUNCTIONS:

4. Load the fixed screen address into a LCD command.
5. Output new screen address command to the LCD module.
6. Delay (40 uSec.) for LCD module to process character.
7. Fetch the last character input from KBD.
8. Send the character to the LCD screen.
9. Delay (40 uSec.) for LCD module to process character.

WRITE ROUTINE

1, 2, 3

FUNCTIONS:

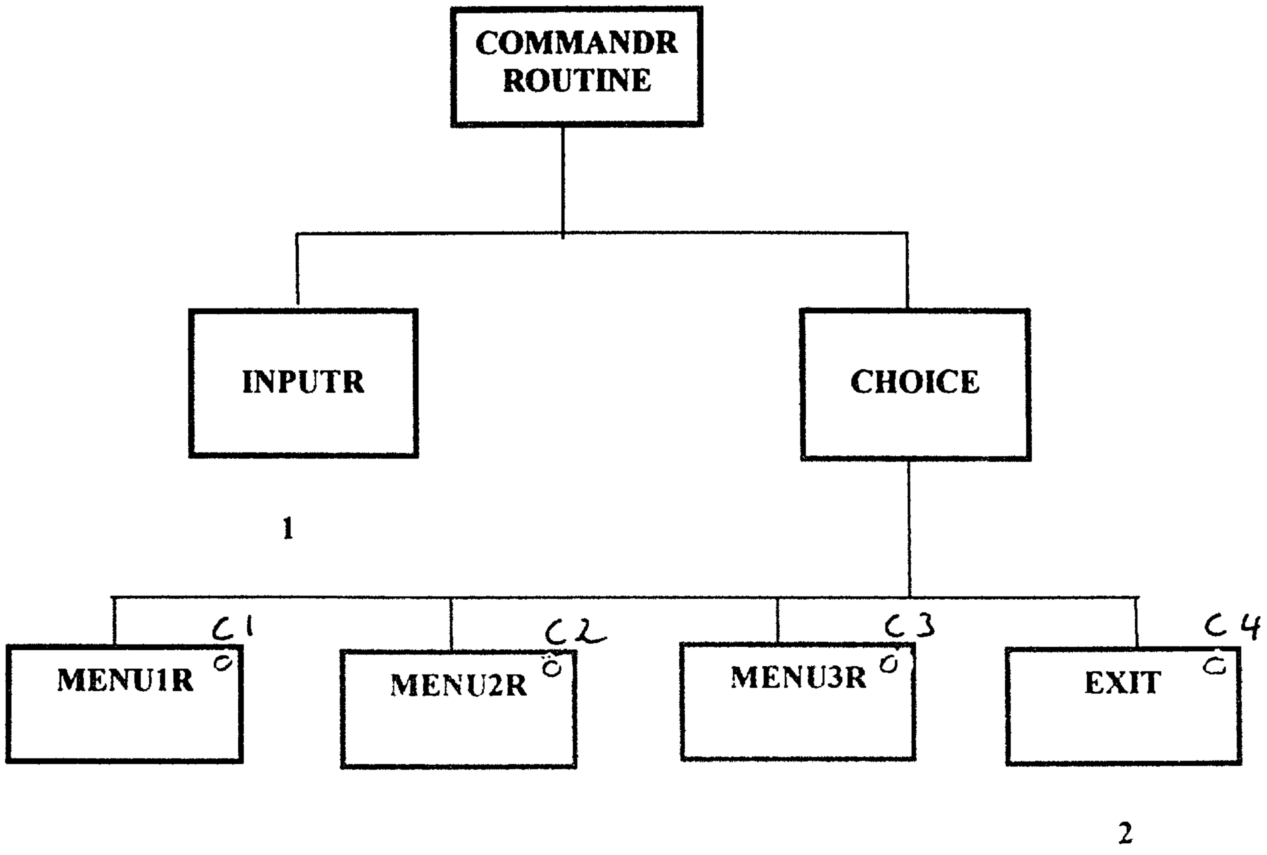
1. Character to be displayed passed as a parameter via accA, and stored in TEMP1.
2. Send character to LCD screen.
3. Delay to allow LCD module to process character.

WRITEHEX

1, 2, 3, 4, 5, 6

FUNCTIONS:

1. Value to be displayed passed via accA, and stored in TEMP2.
2. Convert upper 4-bits of value to be displayed to an ASCIIcharacter.
3. Write character to screen.
4. Extract lower 4-bits of parameter
5. Convert to an ASCII character.
6. Write character to screen.



FUNCTIONS:

1. INPUTR subroutine (Input the menu choice).

2. The exit to main routine detected. (NB, for test purposes only)

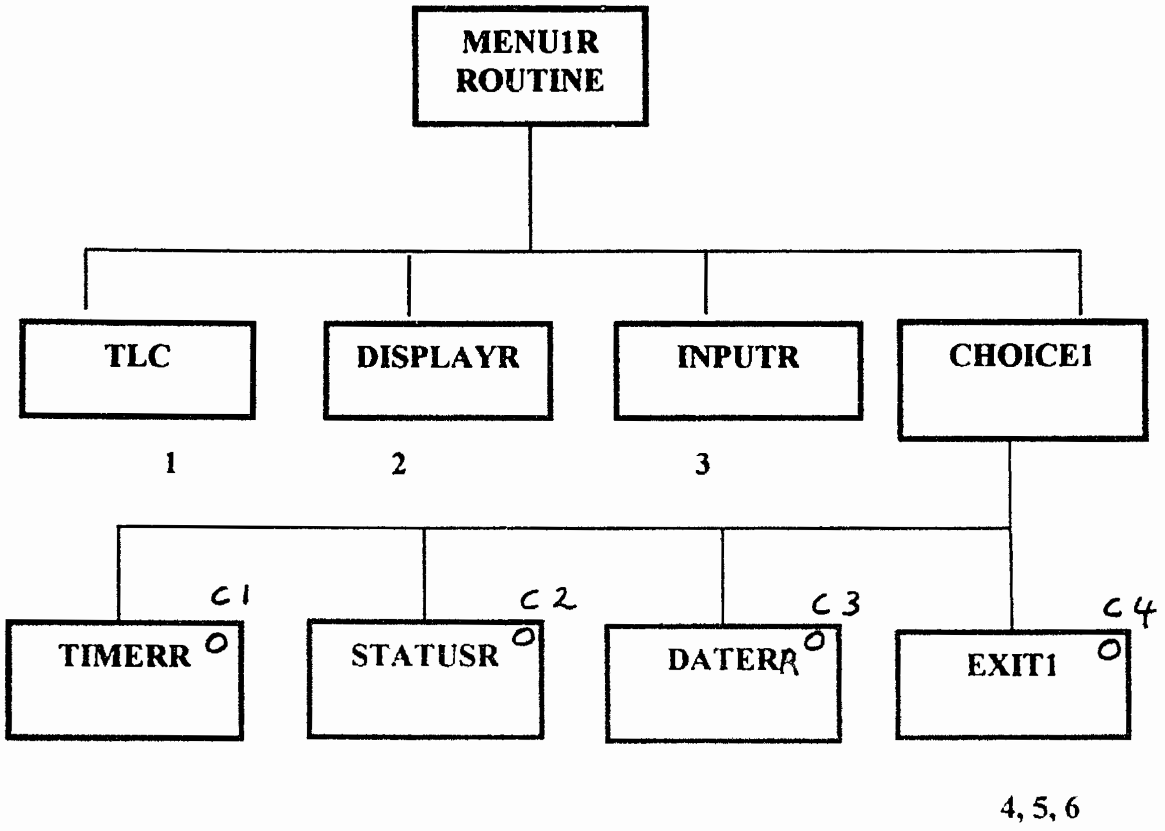
CONDITIONS:

C1. Key number '1' pressed on the hexadecimal keypad.

C2. Key number '2' pressed on the hexadecimal keypad.

C3. Key number '3' pressed on the hexadecimal keypad.

C4. Key number 'E' pressed on the hexadecimal keypad.



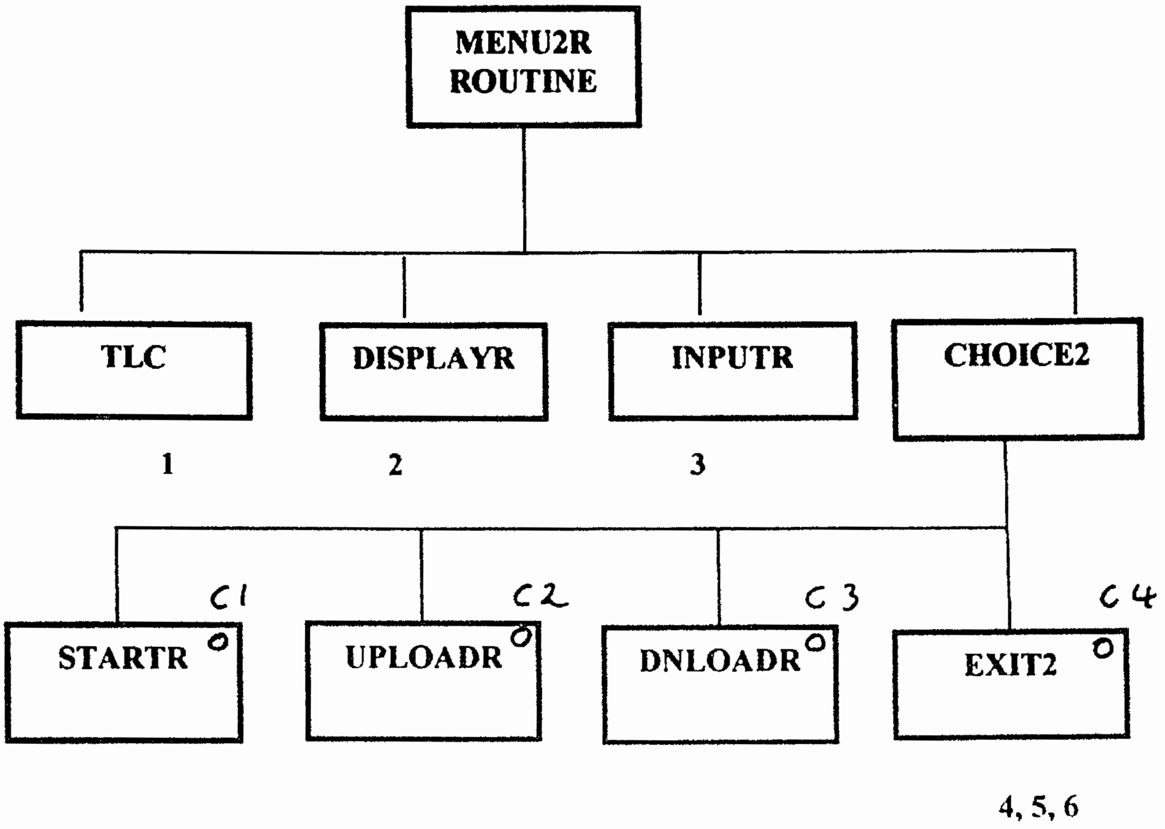
FUNCTIONS:

1. TLC subroutine (Move cursor to TLC of screen).
2. DISPLAYR subroutine (Display MEMU1 on LCD screen).
3. INPUTR subroutine (Input choice of information to be displayed, or choose to exit).

4. TLC subroutine (Move cursor to TLC of screen).
5. DISPLAYR subroutine (Display MEMU on LCD screen)
6. Return to COMMANDR routine.

CONDITIONS:

C1.	Key number `1' pressed on the hexadecimal keypad.
C2.	Key number `2' pressed on the hexadecimal keypad.
C3.	Key number `3' pressed on the hexadecimal keypad.
C4.	Key number `E' pressed on the hexadecimal keypad.



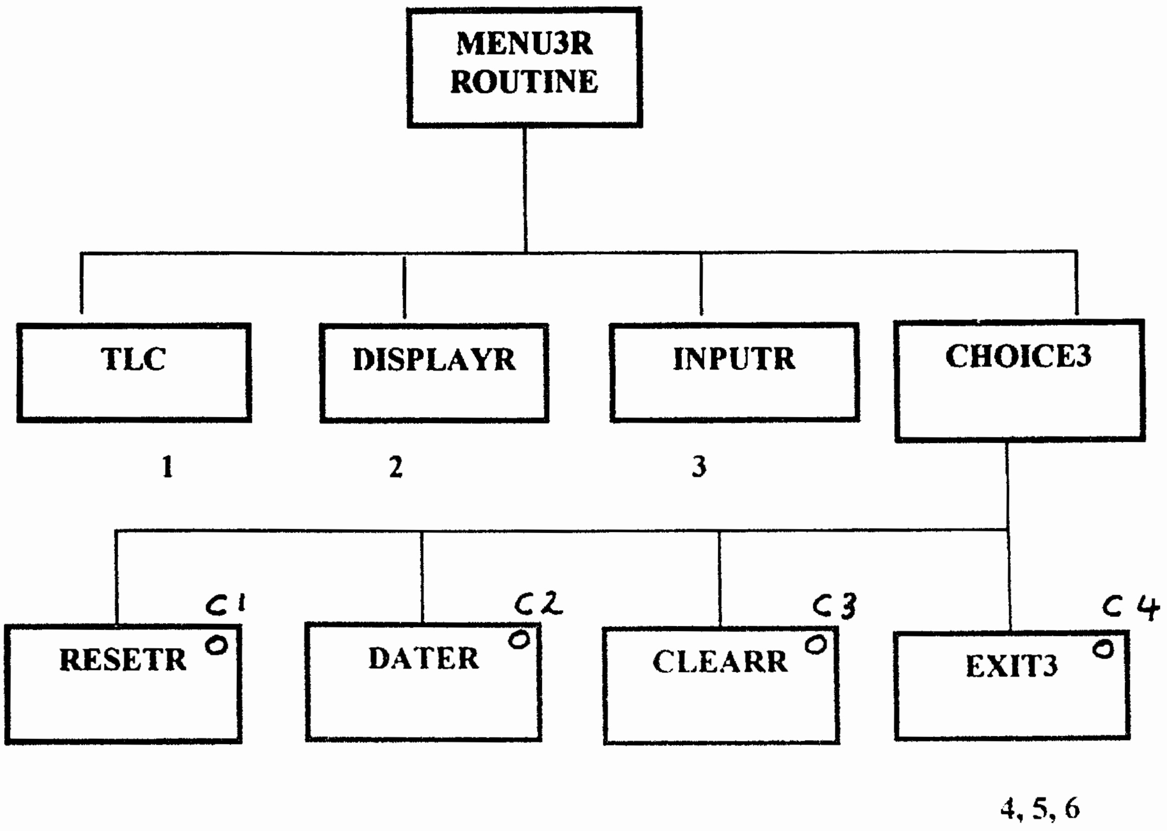
FUNCTIONS:

1. TLC subroutine (Move cursor to TLC of screen).
2. DISPLAYR subroutine (Display MEMU1 on LCD screen).
3. INPUTR subroutine (Input choice of information to be displayed, or choose to exit).

4. TLC subroutine (Move cursor to TLC of screen).
5. DISPLAYR subroutine (Display MEMU on LCD screen).
6. Return to COMMANDR routine.

CONDITIONS:

C1.	Key number `1' pressed on the hexadecimal keypad.
C2.	Key number `2' pressed on the hexadecimal keypad.
C3.	Key number `3' pressed on the hexadecimal keypad.
C4.	Key number `E' pressed on the hexadecimal keypad.



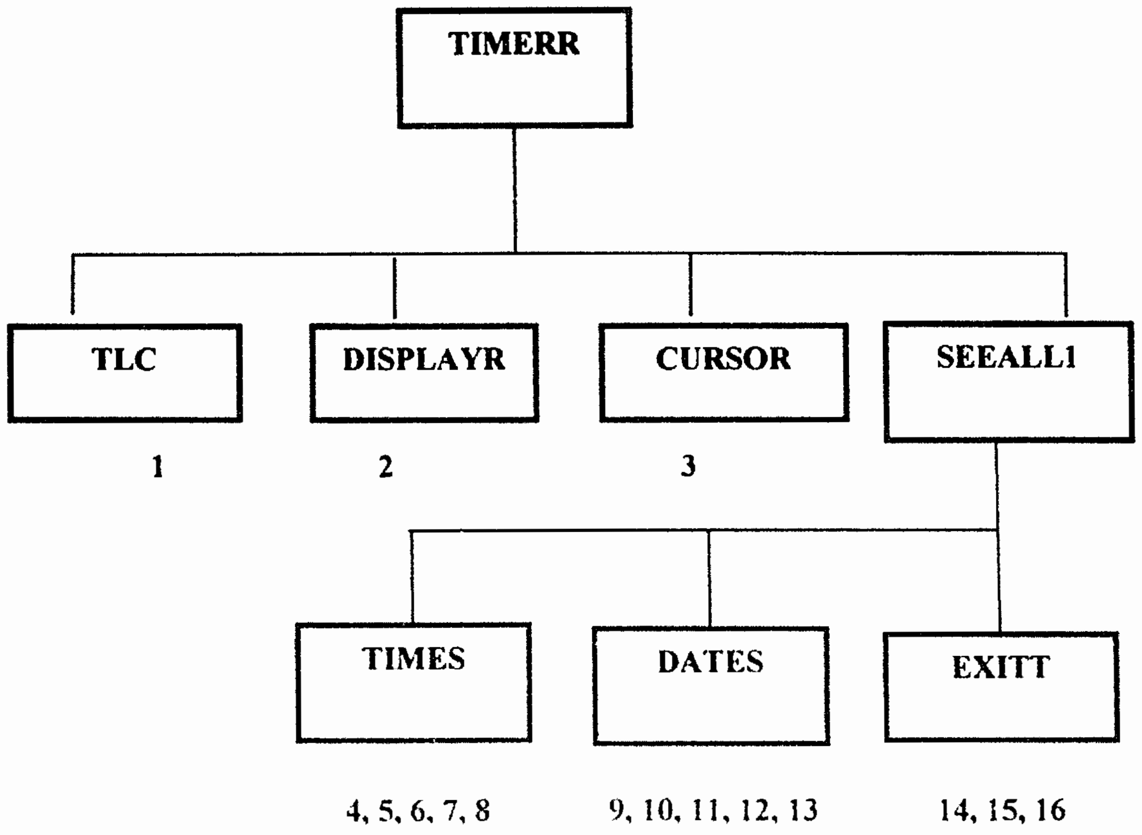
FUNCTIONS:

1. TLC subroutine (Move cursor to TLC of screen)
2. DISPLAYR subroutine (Display MEMU1 on LCD screen).
3. INPUTR subroutine (Input choice of information to be displayed, or choose to exit).

4. TLC subroutine (Move cursor to TLC of screen).
5. DISPLAYR subroutine (Display MEMU on LCD screen).
6. Return to COMMANDR routine.

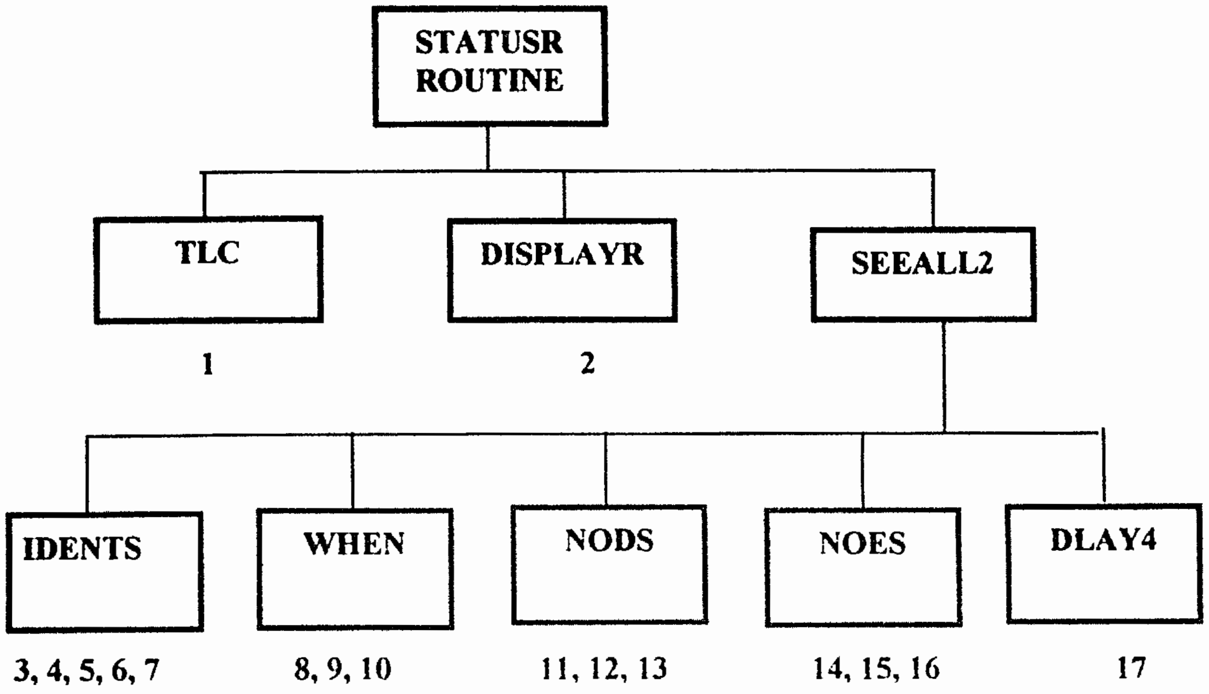
CONDITIONS:

C1.	Key number '1' pressed on the hexadecimal keypad.
C2.	Key number '2' pressed on the hexadecimal keypad.
C3.	Key number '3' pressed on the hexadecimal keypad.
C4.	Key number 'E' pressed on the hexadecimal keypad.



FUNCTIONS:

1.	TLC subroutine (Move cursor to the TLC of the screen).
2.	DISPLAYR subroutine (Display MSG1_1 on LCD screen).
3.	CURSOR subroutine (Move cursor to a new position).
4.	Display day.
5.	space.
6.	Display month.
7.	space.
8.	Display year.
9.	Display hours.
10.	space.
11.	Display minutes.
12.	space.
13.	Display seconds.
14.	Input character from KBD.
15.	Check for exit key.
16.	Return to MENU1.



FUNCTIONS:

1. TLC subroutine (Move cursor to TLC of screen).
2. DISPLAYR subroutine (Display MSG1-2 on LCD screen).

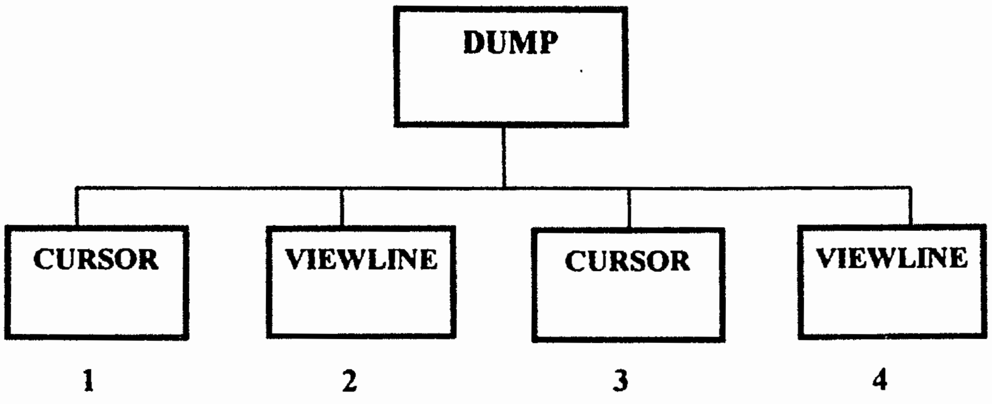
3. CURSOR subroutine (Move cursor to a new position)
4. Get first byte of IDENT
5. WRITE subroutine (Display byte).
6. Get second byte of IDENT.
7. WRITE subroutine (Display byte).

8. CURSOR subroutine (Move cursor to a new position)
9. Get date information.
10. WRITE subroutine (Display date).

11. CURSOR subroutine (Move cursor to a new position).
12. Get the number of data items
13. WRITE subroutine (Display the number of data items).

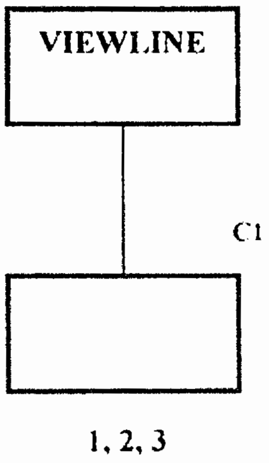
14. Move cursor to a new position.
15. Get the number of events recorded.
16. WRITE subroutine (Display the number of events recorded).

17. Delay time to view status information.



FUNCTIONS:

1. CURSOR subroutine (Move cursor to a new position).
2. VIEWLINE subroutine (Viewline of data).
3. CURSOR subroutine (Move cursor to a new position.
4. VIEWLINE subroutine (Viewline of data).

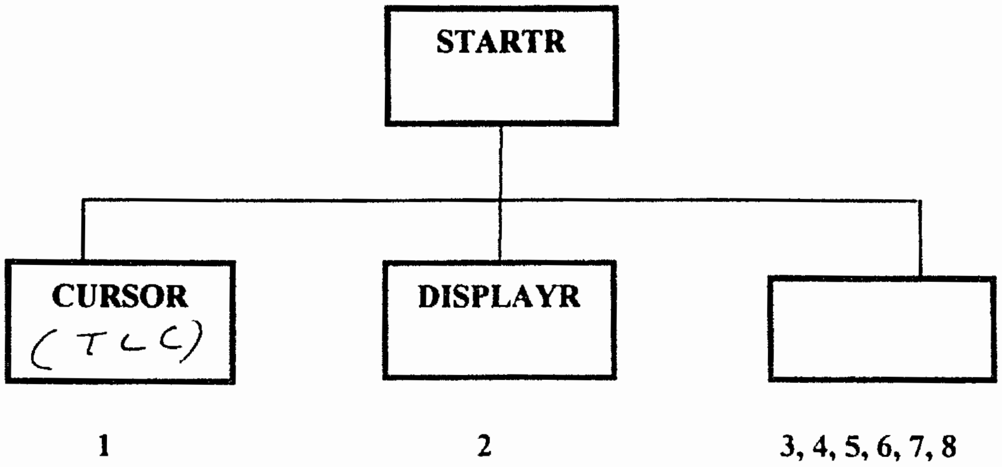


CONDITIONS:

C1. Display 5 bytes of data per line.

FUNCTIONS:

1. Fetch next byte of data.
2. WRITEHEX subroutine (Write two hexadecimal characters).
3. WRITE subroutine (Write a space character).

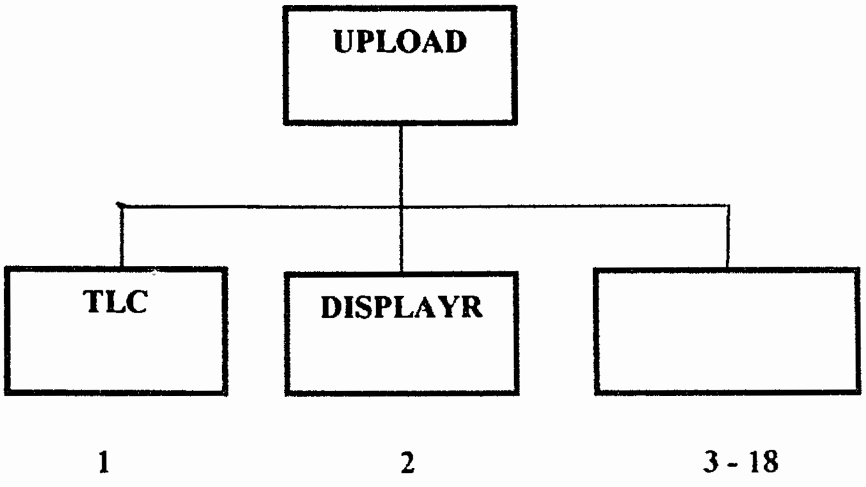


FUNCTIONS:

1. CURSOR subroutine (Move cursor to the TLC of the screen).

2. DISPLAYR subroutine (Display MSG2 1 on the LCD screen).

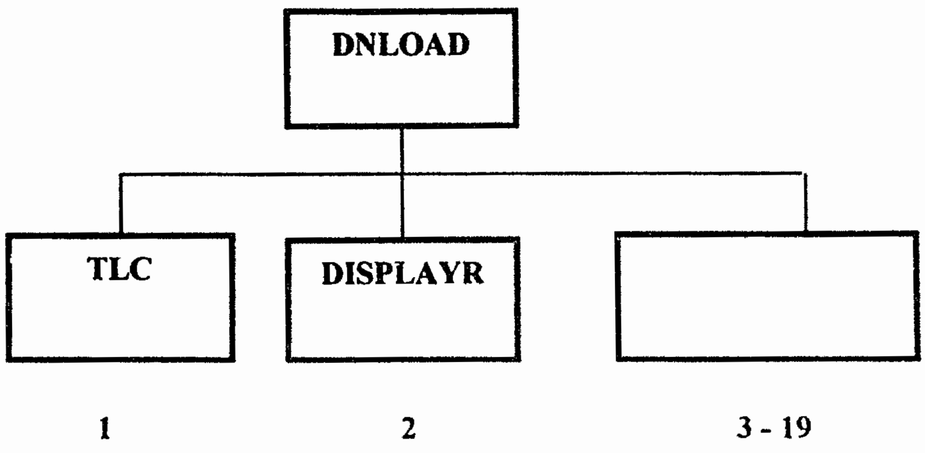
3. Enable SCI transmitter.
4. Output a start of block marker (*).
5. Output a start of of test date and time.
6. Output the end of block marker (#).
7. Delay to indicate routine has been entered.
8. Return to MENU2 routine.



FUNCTIONS:

1. Move cursor to TLC of the LCD screen.
2. Display MSG2_2 on the LCD screen.

3. Clear counter values ready for use
4. Enable SCI receiver.
5. Wait and detect start of block marker (*)
6. Position cursor.
7. Input IDENT from serial port.
8. Write IDENT to LCD screen.
9. Input IDENT+1 from serial port.
10. Write IDENT+1 to LCD screen.
11. Position cursor.
12. Input date from serial port.
13. Write date to LCD screen.
14. Input and store data
15. Detect end of data marker (@)
16. Input and store events.
17. Detect end of block marker (#).
18. Delay for testing program.
19. Return to MENU2 routine.

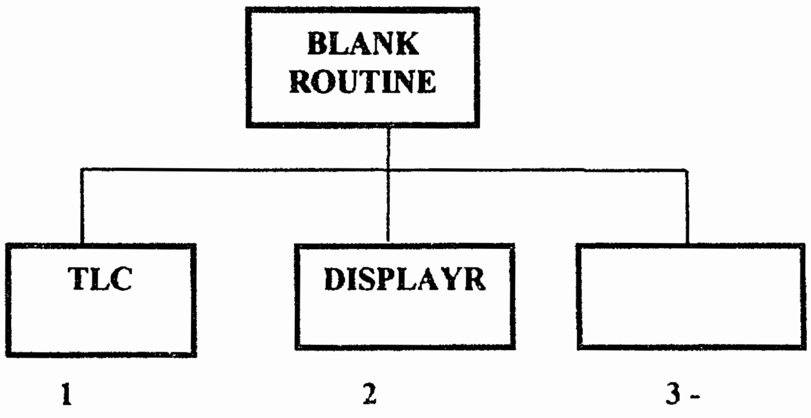


FUNCTIONS:

1. TLC subroutine (Move cursor to the TLC of the screen).

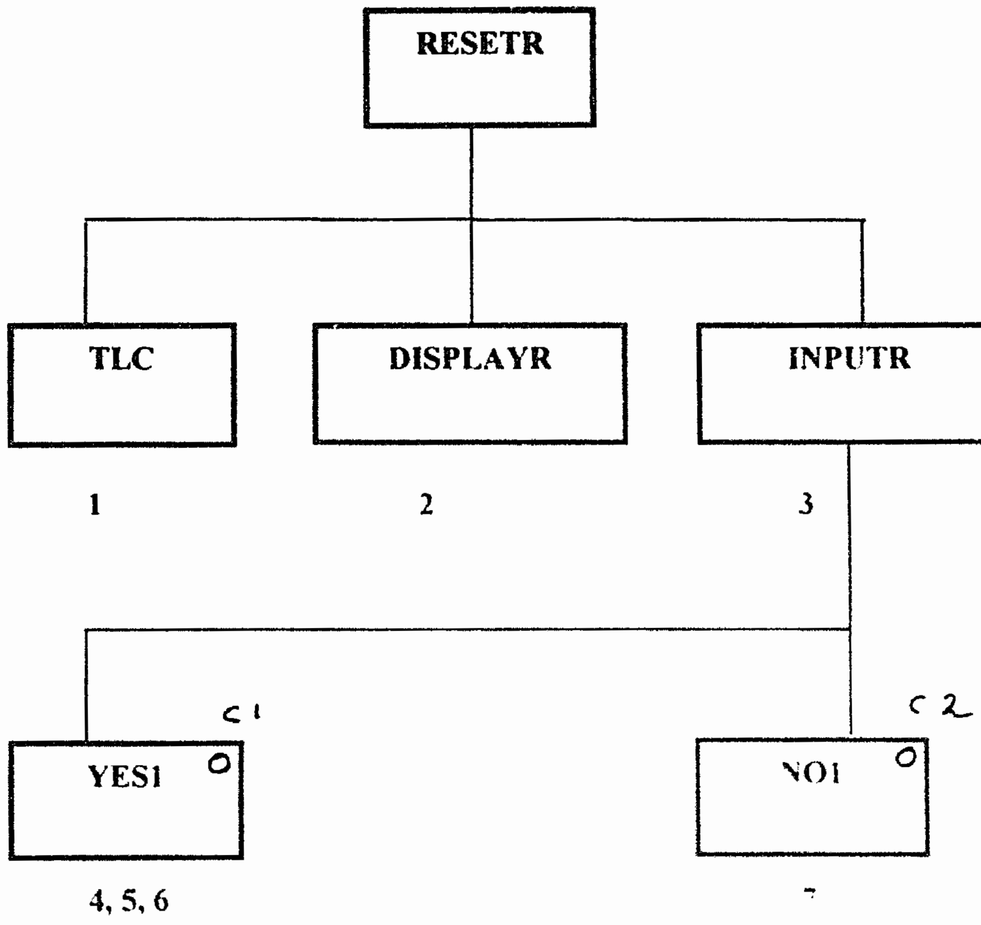
2. DISPLAYR subroutine (Display MSG2 3 on the LCD screen).

3. Enable the SCI transmitter.
4. Output a start of block marker (*)
5. Move the cursor to a new position
6. Output IDENT to the serial port.
7 Write IDENT to the LCD screen.
8. Move the cursor to a new position.
9. Output IDENT+1 to the serial port.
10. Write IDENT+1 to the LCD screen.
11. Move the cursor to a new position.
12. Output the date and time to the serial port.
13. Write the date and time to the LCD screen.
14. Output the stored data to the serial port.
15. Output an end of data marker (@).
16. Output the recorded event times.
17. Output an end of block marker (#).
18. Delay to indicate that this routine has been entered.
19. Return to the MENU2 routine.



FUNCTIONS:

1. TLC subroutine (Move cursor to the TLC of the LCD screen).
2. DISPLAYR routine (Display a blank screen (BLANK))
3 Delay used to acknowledge a response to a key press



CONDITIONS:

C1	The number '1' key must be pressed
C2	The number '2' key must be pressed

FUNCTIONS:

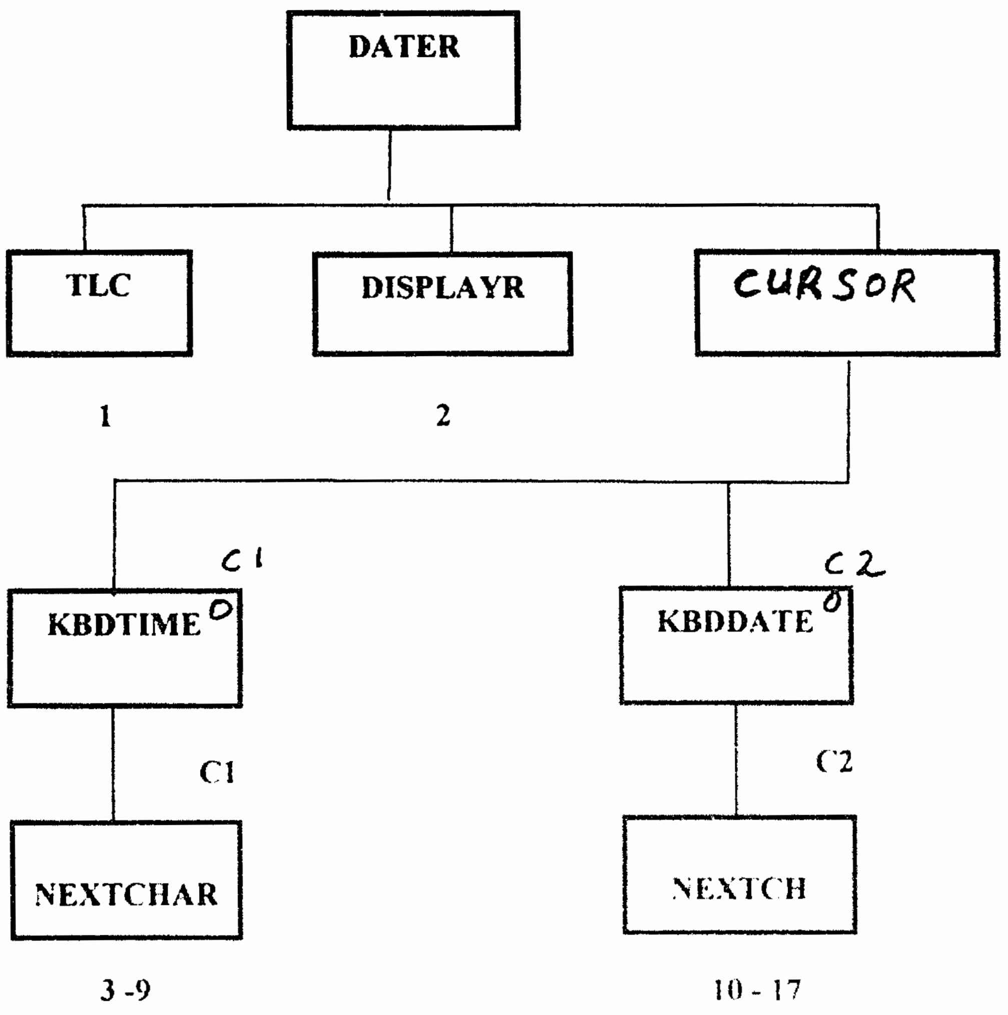
1 TLC subroutine (Move the cursor to the TLC of the LCD screen).

2. DISPLAYR routine (Display MSG3_1 on the LCD screen).

3. INPUTR subroutine (Input a response to the 'yes' / 'no' message).

4. Blank the LCD screen for a couple of seconds.
5. Load the SP with the original 'top of the stack' value.
6. Jump to the MAIN routine to reset the DTD system.

7. Return to the MENU2 routine.



CONDITIONS:

C1. 6 decimal digits need to be entered

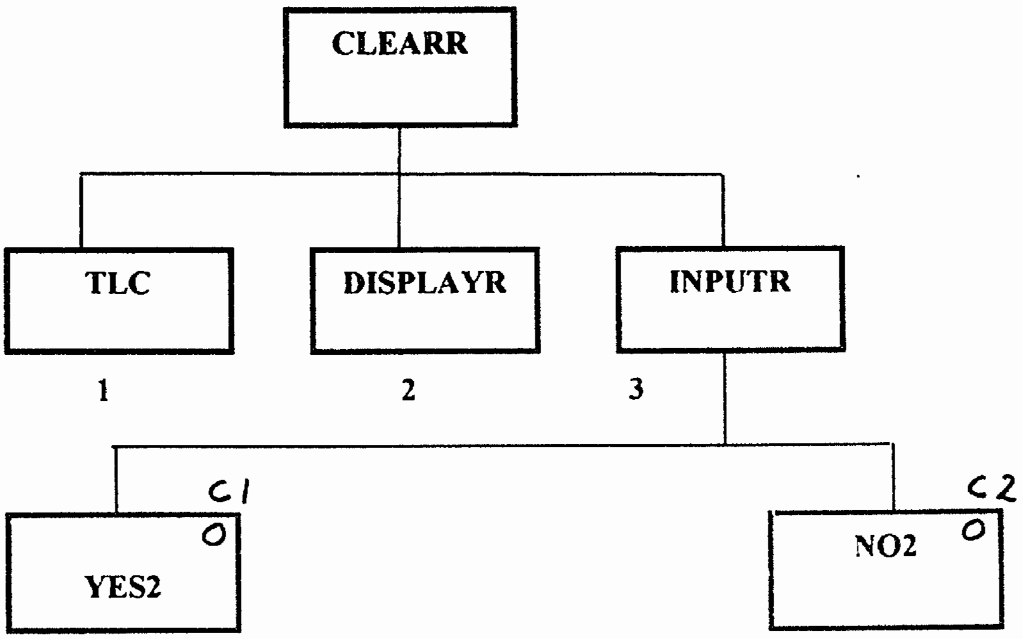
C2 6 decimal digits need to be entered.

FUNCTIONS:

1 Move the cursor to the TLC of the LCD screen.
2. Display MSG3_2 on the LCD screen.

3. Input next character.
4. Accept only BCD values.
5. Convert to ASCII character.
6. Store character at TEMP1.
7. Write to LCD screen.
8. Check for second character. If yes write a space to LCD.
9. Check for fourth character. If yes write a space to LCD.

10. Input next character.
11. Accept only BCD values.
12. Convert to ASCII character.
13. Store character at TEMP1.
14. Write to LCD screen.
15. Check for second character. If yes write a space to LCD.
16. Check for fourth character. If yes write a space to LCD.
17. Wait for an `E' key-press.



CONDITIONS:

C1. Key number '1' needs to be pressed

C2. Key number '2' needs to be pressed

FUNCTIONS:

1. TLC subroutine (Move the cursor to the TLC of the LCD screen)
2. DISPLAYR routine (Display MSG3_2 on the LCD screen)
3. INPUTR subroutine (Input next character).

4. Clear the LCD screen.
5. Clear all buffers and counters.
6. Return to MENU3R routine.

7. Return to MENU3R routine.

B.3.

The Data Transfer Device

program listing

```
  BTEXT

        DTDPROG.ASM              ( 20 - 6 - 1993 )

        The LCD interface uses bits 4, 5 & 6 of port
        `A' for control purposes and bits 2, 3, 4 & 5
        of port `D'for the transfer of data.

        Note, Port `A' bits 4, 5 & 6 used
              for LCD controls `R/W', RS & `E'.
              Port `D' bits 2, 3, 4 & 5 are used
              for the four LCD data bits.

        The hex keypad interface is initialised, the
        input capture flags are cleared ready to accept
        a data available signal from the keyboard.

              Note, Port `A' bit 3 (IC1) is used
              as the KBD data available input.
              Port `E' bits 4, 5, 6 and 7 are
              for the KBD data input lines

  ETEXT

**** STORAGE OF TMD INFORMATION ****

        ORG  $C000

DATEBUF   FCC   "            " ; ASCII date and time buffer
BCDBUF        RMB   6            ; DATE BCD buffer
BUFFER        RMB   1000         ; DATA buffer
EVENTB        RMB   256          ; EVENTS buffer: time elapsed values

IDENT         FDB   $0000      ; storage for TMD identification

*** PARAMETERS USED TO INITIALISE    LCD MODULE ****

TIME   FCB  $30          ; delay times used during
     FCB  $10            ; initialisation
     FCB  $01
     FCB  $01
     FCB  $01
     FCB  $10
     FCB  $30
     FCB  $01
```

```
INSTRUCT
      FCB  $30              ; instructions used to
      FCB  $30              ; initialise the LCD
      FCB  $30
      FCB  $28              ; function set
      FCB  $28              ; function set   ( 4-bit )
      FCB  $08              ; display off
      FCB  $01              ; display clear
      FCB  $0F              ; display on: cursor & blink

**** MESSAGES   FOR LCD DISPLAY MODULE ****

MENU
       FCC "Press 1, 2 or 3   "
       FCC "   TRANSFER (2)  "
       FCC "   DISPLAY  (1)  "
       FCC "   CHANGE   (3) $"

MENU1
       FCC "DISPLAY          "
       FCC "     status (2)  "
       FCC "       time (1)  "
       FCC "       data (3) $"

MENU2
       FCC "TRANSFER         "
       FCC " upload data (2)  "
       FCC " date & time (1)  "
       FCC " dnload data (3) $"

MENU3
       FCC "CHANGE           "
       FCC " time &  date (2)  "
       FCC " reset ststem (1)  "
       FCC " clear   data (3) $"


MSG1_1
       FCC " TIME            "
       FCC " DATE            "
       FCC "       hh:mm:ss  "
       FCC "       dd-mm-yy $"


MSG1_2
       FCC "identification =   "
       FCC "items of data =    "
       FCC "test date= dd-mm-yy "
       FCC "no. of events =   $"
```

```
MSG1_3
        FCC "DATA  b, f, 1, 9 & e"
        FCC "   00 00 00 00 00   "
        FCC "                    "
        FCC "   00 00 00 00 00  $"

MSG2_1
        FCC "START TEST          "
        FCC "   date  =          "
        FCC "   baud  =          "
        FCC "   time  =         $"

MSG2_2
        FCC "UPLOADING           "
        FCC " ident  =           "
        FCC "                    "
        FCC " date   =          $"

MSG2_3
        FCC "DOWNLOADING         "
        FCC " ident  =           "
        FCC "                    "
        FCC " date              $"

MSG3_1
        FCC " RESET THE SYSTEM ? "
        FCC " are you sure (1)=Y "
        FCC "                    "
        FCC "            (2)=N$"

MSG3_2
        FCC "TIME                "
        FCC "DATE                "
        FCC "    Enter hh:mm:ss  "
        FCC "    Enter dd-mm-yy $"

MSG3_3
        FCC "CLEAR LAST RESULTS? "
        FCC " are you sure (1)=Y "
        FCC "                    "
        FCC "            (2)=N$"
```

```
ERROR1
        FCC "****** ERROR1 ******"
        FCC " transmission error "
        FCC "                    "
        FCC "                   $"


ERROR2
        FCC "****** ERROR2 ******"
        FCC "keyboard input error"
        FCC "                    "
        FCC "                   $"


BLANK
        FCC "                    "
        FCC "                    "
        FCC "                    "
        FCC "                   $"

          ****  COUNTERS, POINTERS  AND VARIABLES  ****
                   ****  USED BY THIS PROGRAM  ****

   ORG $0000

DATA        FCB  $00          ; raw data in binary form

COUNT       FCB  $00          ; count of keyboard characters

AMOUNTD     FDB  $0000        ; amount of data items

AMOUNTE     FDB  $0000        ; amount of event items

DPTRMAX     FDB  $0000        ; end of BUFFER pointer

EPTRMAX     FDB  $0000        ; end of EVENTB pointer

HEXBUF      FCB  $20          ; a 4 character ASCII buffer
            FCB  $20
            FCB  $20
            FCB  $20

BUFFPTR     FDB    BUFFER     ; pointer used to store data
EVENTPTR    FDB    EVENTB     ; pointer used to store time of events

ONEMIN      FDB    $0000      ; 1 minute timeout counter

TEMP1       FCB  $00     ; temporary storage for creation of upper bits
TEMP2       FCB  $00
```

```
**** FIXED PARAMETERS MEANT    FOR EEPROM ****

  ORG    $CC00

DATAA        FDB   BUFFER
EVENTS       FDB   EVENTB
DATE FDB     DATEBUF

BUFFMAX FDB      BUFFER        + 1000
PTREMAX FDB      EVENTB        +  256
DATEMAX FDB      DATEBUF +  13

BEGCODE FCB      $2A           ; start of block marker '*'
ENDCODE FCB      $23           ; end    of block marker '#'

TILIMIT FDB $0100              ; timeout value: number of TOFs



**** THE TIMER SUB SYSTEM REGISTERS ****

TCTL2        EQU   $1021
TFLG1        EQU   $1023
TMSK2        EQU   $1024
TFLG2        EQU   $1025
TCNT         EQU   $100E

**** THE SCI SUB SYSTEM REGISTERS ****

BAUD         EQU   $102B
SCCR1         EQU   $102C
SCCR2         EQU   $102D
SCSR         EQU   $102E
SCDR         EQU   $102F

**** THE PARALLEL PORT REGISTERS ****

PORTA        EQU   $1000
PORTB        EQU   $1004
PORTC        EQU   $1003
PORTD        EQU   $1008
PORTE        EQU   $100A

**** THE DATA DIRECTION REGISTERS ****

DDRD         EQU   $1009
PACTL        EQU   $1026
```

**** MAIN ROUTINE FOR DTD PROGRAM ****

```
  ORG  $D000

MAIN
   NOP
   JSR  INIT1R                 ; initialise the sub systems
   NOP
   JSR  INIT2R                 ; initialise LCD 4-bit interface
   NOP
   JSR  INIT3R                 ; initialise the keypad interface
   NOP
   JSR  INIT4R                 ; clear all buffers, counters, pointers etc.
   nop
   LDX  #MENU                  ; 1st screen menu
   JSR  DISPLAYR               ; output menu messages
   NOP
AGEN
   JSR  COMMANDR               ; input commands from KBD routine
   NOP
   WAI

   NOP
   NOP
   NOP

**** INITIALISATION ROUTINES        ****

INIT1R
       NOP                     ; initialise microcontroller SCI sub system
       LDAB #$FF               ; set latch to enable RxD to PD0
       STAB $4000
       NOP
       LDAB #$30               ; set 9600 baud rate
       STAB BAUD
       NOP
       LDAB #$00               ; set SCI for 8 data bits
       STAB SCCR1
       NOP
       LDAB SCCR2              ; disable SCI transmitter, receiver
       ANDB #$03               ; and interrupts
       STAB SCCR2
       NOP
       RTS
```

```
INIT2R
   NOP                          ; ****  8-BIT LCD INTERFACE  ****
   LDAB #$A0                    ; wait 16 milliseconds
   JSR  DELAY
   NOP
   LDX  #TIME
LOOP CPX  #TIME+4               ; fourth instruction been output?
   BEQ  BACK
   NOP
   LDAA 08,X                    ; fetch next instruction
   JSR  OUTPUT
   NOP
   LDAB 00,X                    ; fetch next time value
   JSR  DELAY
   NOP
   INX
   BRA  LOOP
BACK NOP                        ; ****  4-BIT LCD INTERFACE  ****
   CPX  #TIME+8                 ; last instruction been output?
   BEQ  BACK2
   NOP
   LDAA 08,X                    ; fetch next instruction
   STAA TEMP1
   JSR  OUTPUT2                 ; O P top 4 bits
   NOP
   LDAB 00,X                    ; fetch next time value
   JSR  DELAY
   NOP
   INX
   BRA  BACK
   NOP
BACK2 RTS


INIT3R
       LDAA  TFLG1              ;clear all input capture flag bits
       ORAA #$FF
       STAA  TFLG1
       NOP
       LDAA #$10                ;program interface to detect +ve edges
       STAA  TCTL2
       NOP
       RTS
```

```
INIT4R
        NOP                           ; clear all data buffers and DTD variables
        LDAA #00
        LDX   #DATA
CL1     STAA 00,X                     ; clear counters
        INX
        CPX   #HEXBUF
        BLO   CL1
        NOP
        LDAA #$20
        LDX   #HEXBUF
CL2     STAA 00,X                     ; store space characters
        INX
        CPX   #HEXBUF+4
        BLO   CL2
        NOP
        LDAA #00
        LDX   #DATEBUF
CL3     STAA 00,X                     , clear date, data and events buffers
        INX
        CPX   #TIME
        BLO   CL3
        NOP
        RTS

  **** CHOICE OF THE THREE MENUS ROUTINE ****

COMMANDR    NOP                       ; select command from KBD routine
        CLR   DATA
        JSR   INPUTR                  ; i p from KBD
        LDAA DATA
        NOP
CHOICE  CMPA      #$10                        ; display menu choice ?
        BNE   AAAA
        NOP
        JSR   MENU1R
        NOP
        BRA COMMANDR
        NOP
AAAA CMPA #$20                        ; transfer menu choice ?
        BNE   BBBB
        NOP
        JSR   MENU2R
        NOP
        BRA COMMANDR
        NOP
BBBB CMPA #$30                        ; change menu choice ?
        BNE   CCCC
        NOP
        JSR   MENU3R
        NOP
        BRA   COMMANDR
CCCC    NOP
        CMPA #$E0                  ; exit program choice ?
        BNE   COMMANDR
        NOP
EXIT    RTS

**** INPUT  CHARACTER FROM  KBD  ROUTINE  ****

INPUTR
        NOP
CFLAGS  LDAA      TFLG1                 ;clear IC1 flag
        ORAA #$FF
        STAA  TFLG1
        NOP
WAIT    LDAA TFLG1                 ;wait for a data available signal
        ANDA #$04                  ;detect IC1 flag set
        BEQ   WAIT
        NOP
        LDAB  PORTE                ; read KBD input from port 'E'
        STAB  DATA                 ;store data
        NOP
        LDY   #$02
DLAY2   BEQ   OUT2
        LDAB  #$FF                 ; delay for anti-bounce purposes
        JSR   DELAY
        NOP
        DEY
        BRA   DLAY2
        NOP
OUT2    LDAA TFLG1                 ;clear IC1 flag
        ORAA #$FF
        STAA  TFLG1
        NOP
        RTS

**** KBD  CHARACTER  TO ASCII CONVERSION

CONVERT
        NOP
        LDAA DATA                  ;convert hex character to ASCII
        LSRA
        LSRA
        LSRA
        LSRA
        CMPA #$0A                  ;distinguish between 0-9 AND A-F
        BLT   DECIMAL
        ADDA #$07                ;A-F characters only
DECIMAL ADDA    #$30
        STAA  HEXBUF
        NOP
        RTS

**** DISPLAY  KBD  CHARACTER  ROUTINE ****

SHOW
        PSHA
        NOP
        LDAA  #$80               ; jump to a new screen address
        ORAA  #$11               ; new line
        STAA  TEMP1
        JSR     OUTPUT2
        NOP
        LDAB  #$01
        JSR     DELAY
        NOP
        LDX    #HEXBUF           ; display next keypad character
        LDAA  00,X
        STAA  TEMP1
        JSR     SCREEN
        NOP
        LDAB  #$01               ; delay to allow screen processing time
        JSR     DELAY
        NOP
        PULA
        RTS

**** OUTPUT  COMMAND (8-BIT      INTERFACE) ROUTINE  ****

OUTPUT
   LDAB #$3E                     ; set PORTD for O/P
    STAB DDRD
    NOP
    LSRA
    LSRA
    STAA PORTD                   ; PORTD       output instruction
    NOP
    LDAA #$00                    ; RS = 0, R/W = 0, E = 0
    STAA PORTA                   ; PORTA output to control LCD
    LDAA #$40                    ; RS = 0, R/W = 0, E = 1
    STAA PORTA                   ; PORTA output to control LCD
    LDAA #$00                    ; RS = 0, R/W = 0, E =0
    STAA PORTA                    ; PORTA output to control LCD
    NOP
    LDAB #$02                    ; set PORTD for I/P
    STAB DDRD
    RTS
```

**** OUTPUT COMMAND (4-BIT INTERFACE) ROUTINE ****

```
OUTPUT2
   LDAB #$3E              ; set PORTD for O/P
    STAB DDRD
    NOP
    LSLA
    LSLA
    PSHA
    NOP
    LDAA TEMP1            ; fetch next instruction
    LSRA
    LSRA
    STAA PORTD            ; PORTD      output instruction (top bits)
    NOP
    LDAA #$00             ; RS = 0, R/W = 0, E = 0
    STAA PORTA            ; PORTA output to control LCD
    LDAA #$40             ; RS = 0, R/W = 0, E = 1
    STAA PORTA            ; PORTA output to control LCD
    LDAA #$00             ; RS = 0, R/W = 0, E = 0
    STAA PORTA            ; PORTA output to control LCD
    NOP


    PULA
    STAA PORTD            ; PORTD output instruction (lower bits)
    NOP
    LDAA #$40             ; RS = 0, R/W = 0, E = 1
    STAA PORTA            ; PORTA output to control LCD
    LDAA #$00             ; RS = 0, R/W = 0, E = 0
    STAA PORTA            ; PORTA output to control LCD
    NOP
    LDAB #$02             ; set PORTD for I/P
    STAB DDRD
    RTS
```

**** OUTPUT ASCII CHARACTER TO LCD MODULE ROUTINE ****

```
SCREEN   LDAB #$3E                    ; set PORTD for O/P
    STAB DDRD
    NOP
    LSLA
    LSLA
    PSHA
    NOP
    LDAA TEMP1            ; fetch the same instruction
    LSRA
    LSRA
    STAA PORTD            ; PORTD      output data (top bits)
    NOP
    LDAA #$20                ; RS = 1, R/W = 0, E = 0
    STAA PORTA               ; PORTA output to control LCD
    LDAA #$60                ; RS = 1, R/W = 0, E = 1
    STAA PORTA               ; PORTA output to control LCD
    LDAA #$20                ; RS = 1, R/W = 0, E =0
    STAA PORTA               ; PORTA output to control LCD
    NOP
    PULA
    STAA PORTD               ; PORTD      output data (lower bits)
    NOP
    LDAA #$60                ; RS = 1, R/W = 0, E = 1
    STAA PORTA               ; PORTA output to control LCD
    LDAA #$20                ; RS = 1, R/W = 0, E =0
    STAA PORTA               ; PORTA output to control LCD
    NOP
    LDAB #$02                ; set PORTD for I/P
    STAB DDRD
    NOP
    RTS

        **** PROCESSING ASCII CHARACTER DELAY ****

DELAY
    LDAA #$00                ; variable delay routine
XX  INCA
    CMPA #20                 ; 100 micro second perloop
    BNE  XX
    DECB
    BNE  DELAY
    NOP
    RTS

**** OUTPUT A SCREEN MESSAGE TO THE LCD MODULE ROUTINE****

DISPLAYR
    NOP                      ; display full_screen message routine
        LDAA 00,X            ; fetch next character
        STAA TEMP1
        CMPA #$24            ; '$'
        BEQ  LAST
        JSR  SCREEN          ; O/P top 4 bits
      NOP
        LDAB #$01            ; 40 microsecond time value
        JSR  DELAY
        NOP
        INX
        BRA  DISPLAYR
        NOP
LAST    RTS
        **** CONTROL ROUTINE FOR DISPLAY CHOICE ****
```

```
MENU1R
        NOP
        JSR    TLC
        NOP
        LDX    #MENU1
        JSR    DISPLAYR
        NOP
        CLR    DATA
        JSR    INPUTR
        LDAA DATA
        CLR    DATA
        NOP
CHOICE1 CMPA    #$10
        BNE    EEEE
        NOP
        JSR    TIMERR
        NOP
        BRA    MENU1R
        NOP
EEEE    CMPA #$20
        BNE    FFFF
        NOP
        JSR    STATUSR
        NOP
        BRA    MENU1R
        NOP
FFFF    CMPA #$30
        BNE    GGGG
        NOP
        JSR    DATERR
        NOP
        BRA    MENU1R
GGGG NOP
        CMPA #$E0
        BNE    MENU1R
        NOP
EXIT1 JSR    TLC
        NOP
        LDX    #MENU
        JSR    DISPLAYR
        NOP
        RTS
```

****** CONTROL ROUTINE FOR TRANSFER CHOICE ******

```
MENU2R
        NOP
        JSR     TLC
        NOP
        LDX     #MENU2
        JSR     DISPLAYR
        NOP
        CLR     DATA
        JSR     INPUTR
        LDAA DATA
        CLR     DATA
        NOP
CHOICE2 CMPA    #$10
        BNE     HHHH
        NOP
        JSR     STARTR
        NOP
        BRA     MENU2R
        NOP
HHHH CMPA #$20
        BNE     IIII
        NOP
        JSR     UPLOADR
        NOP
        BRA     MENU2R
        NOP
IIII    CMPA #$30
        BNE     JJJJ
        NOP
        JSR     DNLOADR
        NOP
        BRA     MENU2R
JJJJ    NOP
        CMPA #$E0
        BNE     MENU2R
        NOP
EXIT2 JSR       TLC
        NOP
        LDX     #MENU
        JSR     DISPLAYR
        NOP
        RTS
```

**** CONTROL ROUTINE FOR CHANGE CHOICE ****

```
MENU3R
        NOP
        JSR     TLC
        NOP
        LDX     #MENU3
        JSR     DISPLAYR
        NOP
        CLR     DATA
        JSR     INPUTR
        LDAA DATA
        CLR     DATA
        NOP
CHOICE3 CMPA    #$10
        BNE     KKKK
        NOP
        JSR     RESETR
        NOP
        BRA     MENU3R
        NOP
KKKK CMPA #$20
        BNE     LLLL
        NOP
        JSR     DATER
        NOP
        BRA     MENU3R
        NOP
LLLL    CMPA #$30
        BNE     MMMM
        NOP
        JSR     CLEARR
        NOP
        BRA     MENU3R
MMMM            NOP
        CMPA #$E0
        BNE     MENU3R
        NOP
EXIT3 JSR       TLC
        NOP
        LDX     #MENU
        JSR     DISPLAYR
        NOP
        RTS
```

**** LCD MODULE OUTPUT ROUTINES ****

```
TLC
        NOP
        LDAA #$80               ; jump to a new screen address
        STAA TEMP1
        JSR    OUTPUT2
        NOP
        LDAB #$01
        JSR    DELAY
        NOP
        RTS


CURSOR
        NOP                     ; jump to a new screen address
        STAA TEMP1
        JSR    OUTPUT2
        NOP
        LDAB #$01
        JSR    DELAY
        NOP
        RTS


WRITE
        NOP                     ; output a character to the LCD
        STAA TEMP1
        JSR    SCREEN
        NOP
        LDAB #$01
        JSR    DELAY
        NOP
        RTS

WRITEHEX
        NOP                 ; 8-bit  HEXADECIMAL to ASCII conversion
        STAA  TEMP2
        STAA  DATA
        JSR     CONVERT             ; high bits
        LDAA  HEXBUF
        JSR     WRITE       ; write character to LCD
        NOP
        LDAA  TEMP2
        ANDA #$0F                   ; lower 4-bits
        ORAA #$30
        CMPA #$39
        BLS    BCD
```

```
      ADDA #$07
BCD   JSR    WRITE          ; write character to LCD
      NOP
      RTS
    **** DISPLAY  DATE  AND  TIME FROME DATE BUFFER  ****

TIMERR
      NOP
      JSR    TLC
      NOP
      LDX    #MSG1_1
      JSR    DISPLAYR
      NOP
      LDAA #$8A                  ; position cursor on LCD
      JSR    CURSOR
SEEALL1 NOP
TIMES LDAA         DATEBUF      ; write stored time to LCD
      JSR    WRITE
      LDAA DATEBUF+1
      JSR    WRITE
      LDAA #$20
      JSR    WRITE
      LDAA DATEBUF+2
      JSR    WRITE
      LDAA DATEBUF+3
      JSR    WRITE
      LDAA #$20
      JSR    WRITE
      LDAA DATEBUF+4
      JSR    WRITE
      LDAA DATEBUF+5
      JSR    WRITE
      NOP
DATES LDAA         #$9E          ; position cursor on LCD
      JSR    CURSOR
      NOP
      LDAA DATEBUF+6             ; write stored date to LCD
      JSR    WRITE
      LDAA DATEBUF+7
      JSR    WRITE
      LDAA #$20
      JSR    WRITE
      LDAA DATEBUF+8
      JSR    WRITE
      LDAA DATEBUF+9
      JSR    WRITE
      LDAA #$20
      JSR    WRITE
      LDAA DATEBUF+10
```

```
        JSR    WRITE
        LDAA DATEBUF+11
        JSR    WRITE
        NOP
EXITT CLR    DATA               ; accept the 'E' characters only
        JSR   INPUTR
        LDAA DATA
        CMPA #$E0
        BNE   EXITT
        NOP
OUT3  RTS

 **** DISPLAY STATUS INFORMATION RECEIVED FROM TMD ****

STATUSR
        NOP
        JSR    TLC
        NOP
        LDX    #MSG1_2
        JSR    DISPLAYR
SEEALL2 NOP
IDENTS LDAA #$91                ; display IDENT
        JSR    CURSOR
        NOP
        LDAA IDENT
        JSR    WRITE
        LDAA IDENT+1
        JSR    WRITE
        NOP
WHEN  LDAA #$CB                 ; display DATE
        JSR    CURSOR
        NOP
        LDX    #DATEBUF+6
DISPDATE  NOP
        LDAA 00,X
        JSR    WRITE
        INX
        CPX    #DATEBUF+11
        BLS    DISPDATE
        NOP
        LDAA #$20               ; write 2 spaces
        JSR    WRITE
        LDAA #$20
        JSR    WRITE
        NOP
NODS LDAA #$A2                  ; display number of DATA items
        JSR    CURSOR
        NOP
        LDAA AMOUNTD            ; high byte
        JSR    WRITEHEX
        LDAA AMOUNTD+1          ;  low byte
        JSR    WRITEHEX
        NOP
NOES LDAA #$E4                  ; display number of EVENTS
        JSR    CURSOR
        NOP
        LDAA AMOUNTE+1
        LSRA
        JSR    WRITEHEX
        NOP
        LDY    #$E8
DLAY4          BEQ    OUT4
        LDAB  #$FF              ; delay for TEST purposes
        JSR   DELAY
        NOP
        DEY
        BRA    DLAY4
        NOP
OUT4 RTS

     **** DISPLAY THE DATA RECEIVED FROM THE TMD ****

DATERR
        NOP
        JSR    TLC
        NOP
        LDX    #MSG1_3
        JSR    DISPLAYR
        NOP
        LDX    #BUFFER          ; dump 2 lines of data, at a time, to the LCD
        JSR    DUMP
        NOP
TOBACK CLR DATA
        JSR    INPUTR
        LDAA DATA
SEEALL3 NOP
EXITDD CMPA #$E0         ; exit ?
        BEQ    EXITD
        NOP
        CMPA #$F0               ; move forward into data buffer
        BNE    MISSF
FORWARD DEX
        DEX
        DEX
        DEX
        DEX
        JSR    DUMP
        BRA    TOBACK
```

```
MISSF NOP
      CMPA #$B0              ; move backward into data buffer
      BNE  MISSB
BACKWARD LDY   #$0000
MINUS      DEX
      INY
      CPY  #$000E
      BLS  MINUS
      JSR  DUMP
      BRA  TOBACK
MISSB NOP
      CMPA #$10              ; move to start of data buffer
      BNE  TOBACK
BEGIN LDX  #BUFFER
      JSR  DUMP
      BRA  TOBACK
EXITD NOP
      RTS


DUMP
      NOP                    ; display data on LCD
      LDAA #$97
      JSR  CURSOR            ; first line position
      NOP
      JSR  VIEWLINE
      NOP
      LDAA #$D7
      JSR  CURSOR            ; second line position
      NOP
      JSR  VIEWLINE
      NOP
      RTS

VIEWLINE
      NOP                    ; display one line of data on LCD
      LDY  #$0000
SEEDATA NOP
      LDAA 00,X
      JSR  WRITEHEX          ; view 2 ASCII-HEXADECIMAL
characters
      INX
      LDAA #$20
      JSR  WRITE             ; space character
      INY
      CPY  #$0004
      BLS  SEEDATA
      NOP
      RTS
```

**** DOWNLOAD THE START OF TEST INFORMATION ****

```
STARTR
        NOP
        JSR    TLC
        NOP
        LDX    #MSG2_1
        JSR    DISPLAYR
        NOP
        LDAB #$08                    ; enable SCI transmitter
        STAB SCCR2
        NOP
AAA     LDAB SCSR                    ; output an '*' character
        ANDB #$80
        BEQ    AAA
        LDAA #$2A
        STAA SCDR
        NOP
BBB     LDAB SCSR                    ; output an '*' character
        ANDB #$80
        BEQ    BBB
        LDAA #$2A
        STAA SCDR
        NOP
        LDX    #DATEBUF              ; output to SCI the date and time
CCC     LDAB SCSR
        ANDB #$80
        BEQ    CCC
        LDAA 00,X
        STAA SCDR
        INX
        CPX    #DATEBUF+11
        BLS    CCC
        NOP
DDD     LDAB SCSR                    ; output an '#' character
        ANDB #$80
        BEQ    DDD
        LDAA #$23
        STAA SCDR
        NOP
        LDY    #$40
DLAY6          BEQ    OUT6
        LDAB   #$FF                  ; delay for TEST purposes
        JSR    DELAY
        NOP
        DEY
        BRA    DLAY6
        NOP
OUT6    RTS
```

****** UPLOAD THE TMD INFORMATION ******

```
UPLOADR
        NOP                         ; upload test data from TMD
        JSR    TLC
        NOP
        LDX    #MSG2_2
        JSR    DISPLAYR
        NOP
        CLR    AMOUNTD              ; clear data  counter
        CLR    AMOUNTD+1
        CLR    AMOUNTE              ; clear event counter
        CLR    AMOUNTE+1
        NOP
        LDAB #$04                   ; enable SCI receiver
        STAB  SCCR2
        NOP
EEE     LDAB SCSR                   ; wait for a character
        ANDB #$20
        BEQ    EEE
        LDAA SCDR                   ; input '*'
        CMPA #$2A
        BNE    EEE
        NOP                         ; position cursor for LCD
        LDAA #$9E
        JSR    CURSOR
        NOP
FFF     LDAB SCSR                   ; wait for a character
        ANDB #$20
        BEQ    FFF
        LDAA SCDR                   ; input IDENT
        STAA  IDENT
        JSR    WRITE                ; write to LCD
        NOP
GGG     LDAB SCSR                   ; wait for a character
        ANDB #$20
        BEQ    GGG
        LDAA SCDR                   ; input IDENT + 1
        STAA  IDENT+1
        JSR    WRITE                ; write to LCD
        NOP
        LDX    #DATEBUF             ; input DATE AND TIME
        NOP
        LDAA #$DE                   ; position cursor for LCD
        JSR    CURSOR
        NOP
HHH     LDAB SCSR                   ; wait for a character
        ANDB #$20
        BEQ    HHH
        LDAA SCDR              ; input DATE
        STAA 00,X
        CPX  #DATEBUF+5
        BLS  PASS
        JSR  WRITE             ; write to LCD
PASS    INX
        CPX  #DATEBUF+11
        BLS  HHH
        NOP
        LDX  #BUFFER           ; input DATA
III     LDAB SCSR              ; wait for a character
        ANDB #$20
        BEQ  III
        LDAA SCDR              ; input DATA
        STAA 00,X
        INC  AMOUNTD+1         ; no. of data items
        BCC  NOC
        INC  AMOUNTD           ; high byte of data count
NOC     INX
        CMPA #$40              ; '@'
        BNE  III
        STX  DPTRMAX
        NOP
        LDX  #EVENTB           ; input EVENTS
JJJ     LDAB SCSR              ; wait for a character
        ANDB #$20
        BEQ  JJJ
        LDAA SCDR              ; input EVENTS
        STAA 00,X
        INC  AMOUNTE+1         ; number of events
        INX
        CMPA #$23              ; '#'
        BNE  JJJ
        STX  EPTRMAX
        NOP
        LDY  #$40
DLAY7        BEQ  OUT7
        LDAB  #$FF             ; delay for TEST purposes
        JSR  DELAY
        NOP
        DEY
        BRA  DLAY7
        NOP
OUT7    RTS
```

**** DOWNLOAD THE TMD INFORMATION ****

```
DNLOADR
      NOP
      JSR   TLC
      NOP
      LDX   #MSG2_3
      JSR   DISPLAYR
      NOP
      LDAB #$08              ; enable SCI transmitter
      STAB SCCR2
      NOP
KKK   LDAB SCSR              ; ready for output?
      ANDB #$80
      BEQ   KKK
      LDAA #$2A              ; start of block marker '*'
      STAA SCDR
      NOP
      LDAA #$9E
      JSR   CURSOR
      NOP
LLL   LDAB SCSR              ; ready for output?
      ANDB #$80
      BEQ   LLL
      LDAA IDENT             ; identification (high byte)
      STAA SCDR
      JSR   WRITE
      NOP
MMM   LDAB SCSR              ; ready for output?
      ANDB #$80
      BEQ   MMM
      LDAA IDENT+1           ; identification (low byte)
      STAA SCDR
      JSR   WRITE
      NOP
      LDAA #$DE
      JSR   CURSOR
      NOP
      LDX   #DATEBUF
NNN   LDAB SCSR              ; ready for output?
      ANDB #$80
      BEQ   NNN
      LDAA 00,X              ; output time then date
      STAA SCDR
      CPX   #DATEBUF+5
      BLS   PASS2
      JSR   WRITE
PASS2 INX
      CPX   #DATEBUF+11
```

```
        BLS   NNN
        NOP
        LDX   #BUFFER
OOO     LDAB SCSR                    ; ready for output?
        ANDB #$80
        BEQ   OOO
        LDAA 00,X                    ; output block of data
        STAA SCDR
        INX
        CMPA #$40                    ; '@'
        BNE   OOO
        NOP
        LDX   #EVENTB
PPP     LDAB SCSR                    ; ready for output?
        ANDB #$80
        BEQ   PPP
        LDAA 00,X                    ; output block of event times
        STAA SCDR
        INX
        CMPA #$23                    ; '#'
        BNE   PPP
        NOP
        LDY   #$40
DLAY8         BEQ   OUT8
        LDAB  #$FF                   ; delay for TEST purposes
        JSR   DELAY
        NOP
        DEY
        BRA   DLAY8
        NOP
OUT8    RTS
```

```
**** RESET ROUTINE ( REINITIALISE THE SYSTEM ) ****

RESETR
        NOP
        JSR     TLC
        NOP
        LDX     #MSG3_1             ; output message to LCD screen
        JSR     DISPLAYR
        NOP
XXXX    CLR     DATA                ; input response to 'are you sure' message
        JSR     INPUTR
        LDAA    DATA
        CLR     DATA
        NOP
        CMPA    #$10                ; 'yes'
        BNE     YYYY
YES1    NOP
        JSR     BLANKR
        NOP
ZZZZ    LDS     #$004A              ; NB top of stack for the EVB
        JMP     MAIN


YYYY    CMPA    #$20                ; 'no'
        BNE     XXXX
NO1     NOP
        NOP
        RTS

**** CLEAR THE LCD MODULE SCREEN ****

BLANKR
        NOP
        JSR     TLC
        NOP
        LDX     #BLANK
        JSR     DISPLAYR            ;
        NOP
        LDY     #$40
DLAYA           BEQ     OUTA
        LDAB    #$FF                ; delay for TEST purposes
        JSR     DELAY
        NOP
        DEY
        BRA     DLAYA
        NOP
OUTA    RTS
```

```
    **** INPUT START OF TEST DATE AND TIME VALUE ****

DATER
        NOP
        JSR    TLC
        NOP
        LDX    #MSG3_2
        JSR    DISPLAYR
        NOP
        LDAA #$8A
        JSR    CURSOR              ; position LCD cursor
        NOP
        JSR    KBDTIME             ; input and store time of test
        NOP
        LDAA #$9E
        JSR    CURSOR              ; position LCD cursor
        NOP
        JSR    KBDDATE             ; input and store date of test
        NOP
OUT9 RTS

**** INPUT START TIME OF TEST ****

KBDTIME
        NOP
        LDX    #DATEBUF
        NOP
NEXTCHAR CLR      DATA
        JSR   INPUTR
        NOP
        LDAA DATA                  ; accept BCD characters only
        CMPA #$90
        BHI    NEXTCHAR
        NOP
        JSR    CONVERT             ; BCD to ASCII convertion
        CLR    DATA                ; clear last data input
        LDAA HEXBUF
        STAA TEMP1
        STAA 00,X
        JSR    SCREEN              ; store character on the LCD
        NOP
        INX
        CPX    #DATEBUF+2
        BNE    CONT1
        LDAA #$20
        JSR    WRITE               ; space character
        BRA    NEXTCHAR
        NOP
```

```
CONT1       CPX   #DATEBUF+4
        BNE   CONT2
        LDAA #$20
        JSR   WRITE             ; space character
        BRA   NEXTCHAR
        NOP
CONT2       CPX   #DATEBUF+6
        BEQ   LASTONE
        NOP
        BRA   NEXTCHAR
        NOP
LASTONE RTS
```

**** INPUT DATE OF TEST ****

```
KBDDATE
        NOP
        LDX   #DATEBUF+6
        NOP
NEXTCH      CLR   DATA
        JSR   INPUTR
        NOP
        LDAA DATA               ; accept BCD characters only
        CMPA #$90
        BHI   NEXTCH
        NOP
        JSR   CONVERT           ; BCD to ASCII convertion
        CLR   DATA              ; clear last data input
        LDAA HEXBUF
        STAA TEMP1
        STAA 00,X
        JSR   SCREEN            ; store character on the LCD
        NOP
        INX
        CPX   #DATEBUF+8
        BNE   CONT3
        LDAA #$20
        JSR   WRITE             ; space character
        BRA   NEXTCH
        NOP
CONT3       CPX   #DATEBUF+10
        BNE   CONT4
        LDAA #$20
        JSR   WRITE             ; space character
        BRA   NEXTCH
        NOP
CONT4       CPX   #DATEBUF+12
        BEQ   LAST11
        NOP
        BRA   NEXTCH
        NOP
LAST11        CLR   DATA        ; accept the `E' characters only
        JSR   INPUTR
        LDAA DATA
        CMPA #$E0
        BNE   LAST11
        NOP
        RTS

   **** CLEAR ALL COUNTERS, BUFFERS AND VARIABLES ****

CLEARR
        NOP
        JSR   TLC
        NOP
        LDX   #MSG3_3
        JSR   DISPLAYR          ; output `are you sure' message to LCD
        NOP
UUUU JSR   INPUTR               ; input response from KBD
        LDAA DATA
        CLR   DATA
        NOP
        CMPA #$10               ; `yes'
        BNE   VVVV
YES2  NOP
        JSR   BLANKR
        NOP
WWWW        JSR   INIT4R        ; clear all buffers, counters etc
        NOP
        BRA   EXITC
        NOP
VVVV CMPA #$20                  ; `no'
        BNE   UUUU
NO2   NOP
EXITC RTS

        END
```

APPENDIX C

The Data Processing System

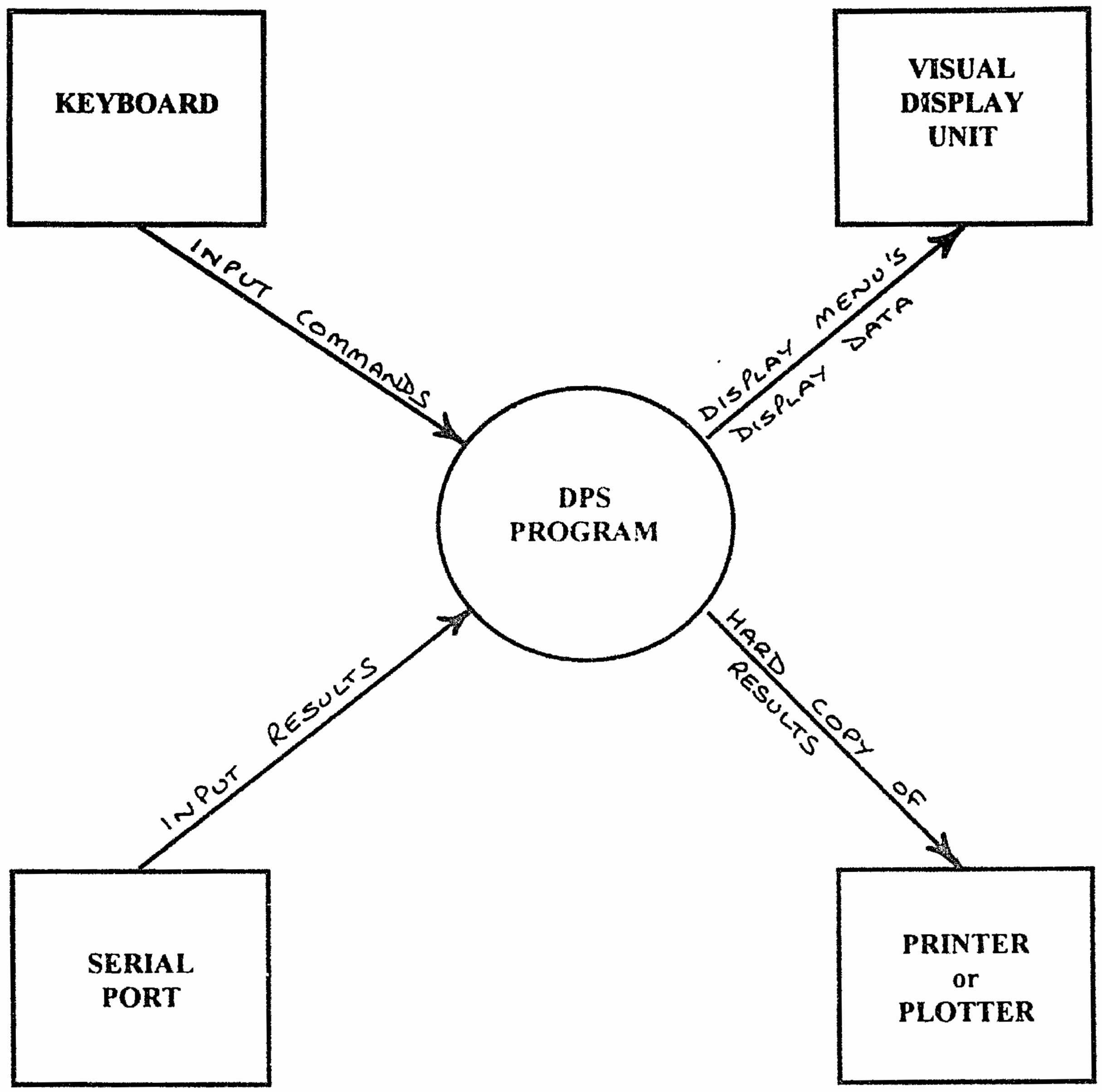
This section contains data flow diagrams, Jackson structure diagrams and program listing for the data processing system.

C.1.

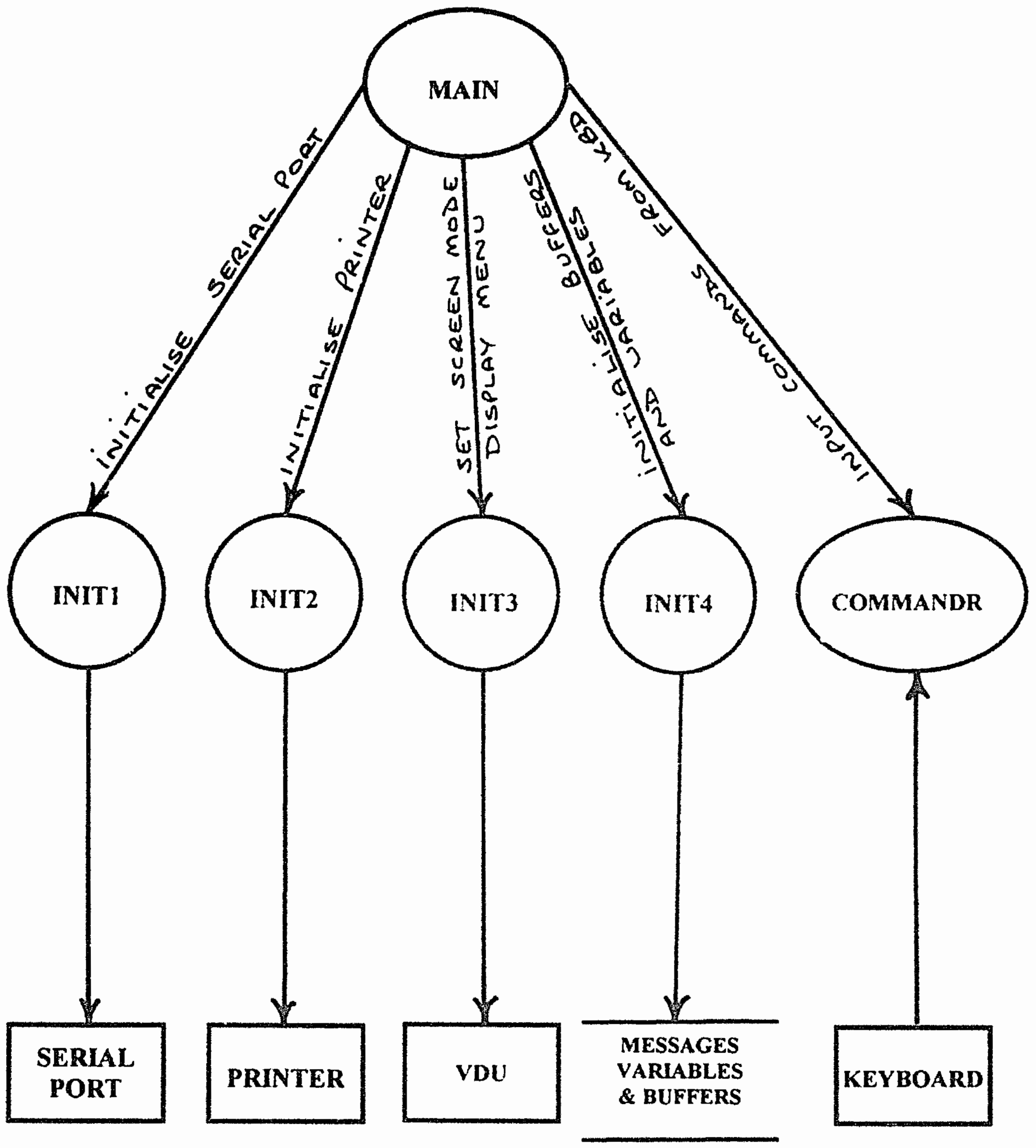
The Data Processing System

data flow diagrams

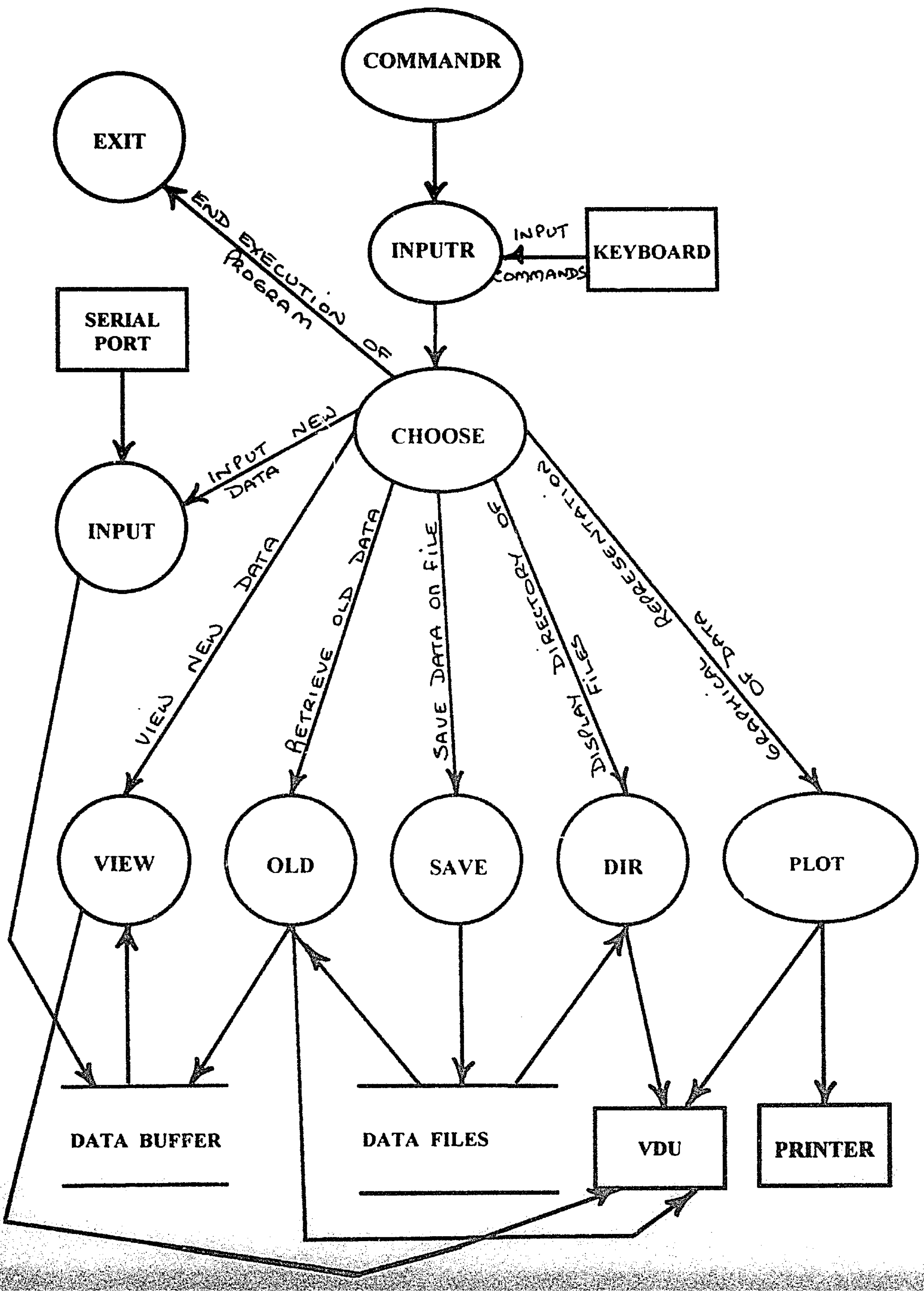
DFD LEVEL 0


KEYBOARD
VISUAL DISPLAY UNIT
DPS PROGRAM
SERIAL PORT
PRINTER or PLOTTER
INPUT COMMANDS
DISPLAY MENU'S
DISPLAY DATA
INPUT RESULTS
HARD COPY OF RESULTS

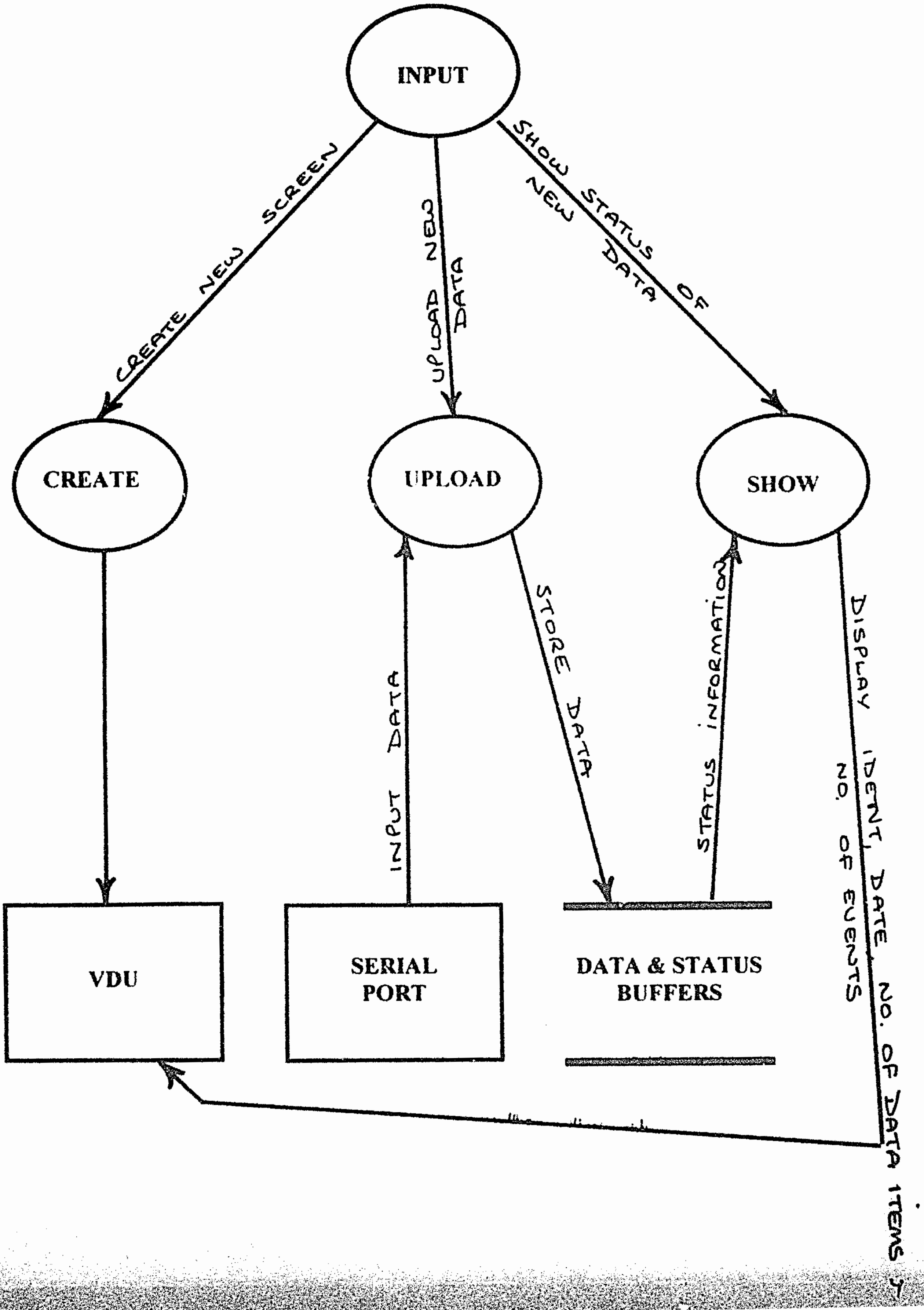
DFD LEVEL 1


MAIN
INITIALISE SERIAL PORT
INITIALISE PRINTER
SET SCREEN MODE
DISPLAY MENU
INITIALISE VARIABLES AND BUFFERS
INPUT COMMANDS FROM KBD
INIT1
INIT2
INIT3
INIT4
COMMANDR
SERIAL PORT
PRINTER
VDU
MESSAGES VARIABLES & BUFFERS
KEYBOARD

DFD LEVEL 2

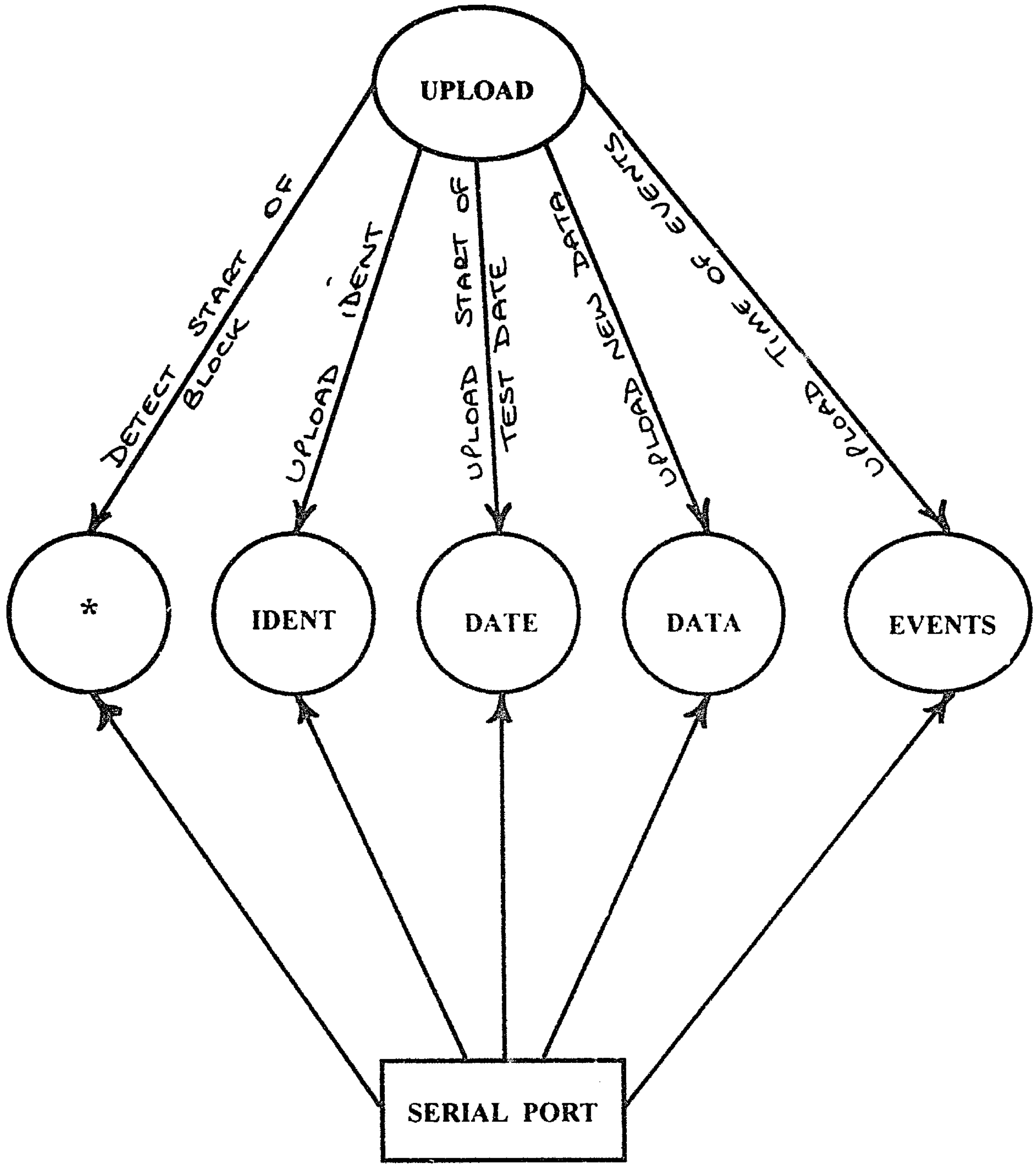

COMMANDR
EXIT
INPUTR
KEYBOARD
INPUT COMMANDS
END EXECUTION OF PROGRAM
SERIAL PORT
CHOOSE
INPUT NEW DATA
INPUT
VIEW NEW DATA
RETRIEVE OLD DATA
SAVE DATA ON FILE
DISPLAY DIRECTORY OF FILES
GRAPHICAL REPRESENTATION OF DATA
VIEW
OLD
SAVE
DIR
PLOT
DATA BUFFER
DATA FILES
VDU
PRINTER

DFD LEVEL 3



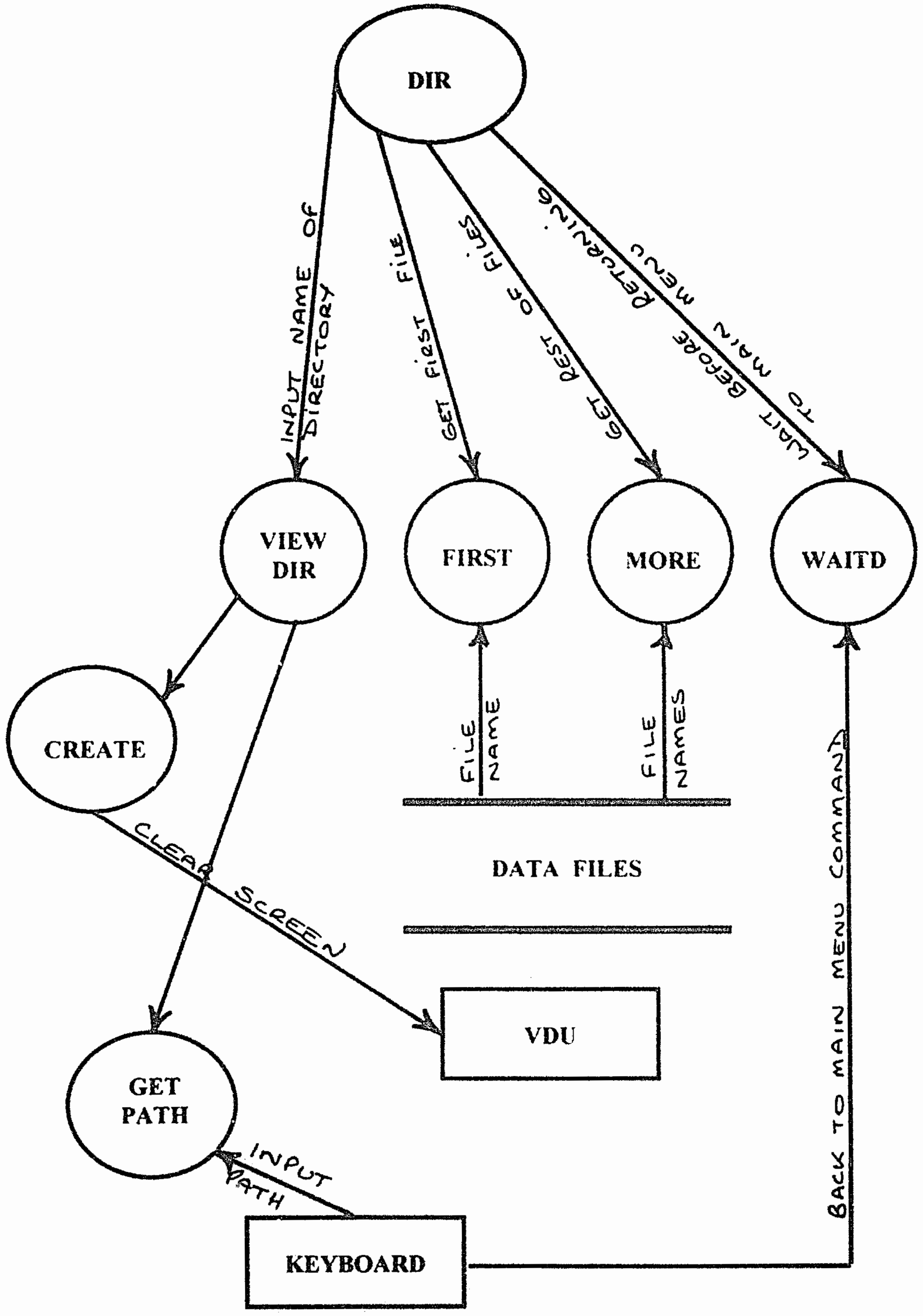
INPUT
CREATE NEW SCREEN
UPLOAD NEW DATA
SHOW STATUS OF NEW DATA
CREATE
UPLOAD
SHOW
INPUT DATA
STORE DATA
STATUS INFORMATION
DISPLAY IDENT, DATE, NO. OF DATA ITEMS, NO. OF EVENTS
VDU
SERIAL PORT
DATA & STATUS BUFFERS

DFD LEVEL 4

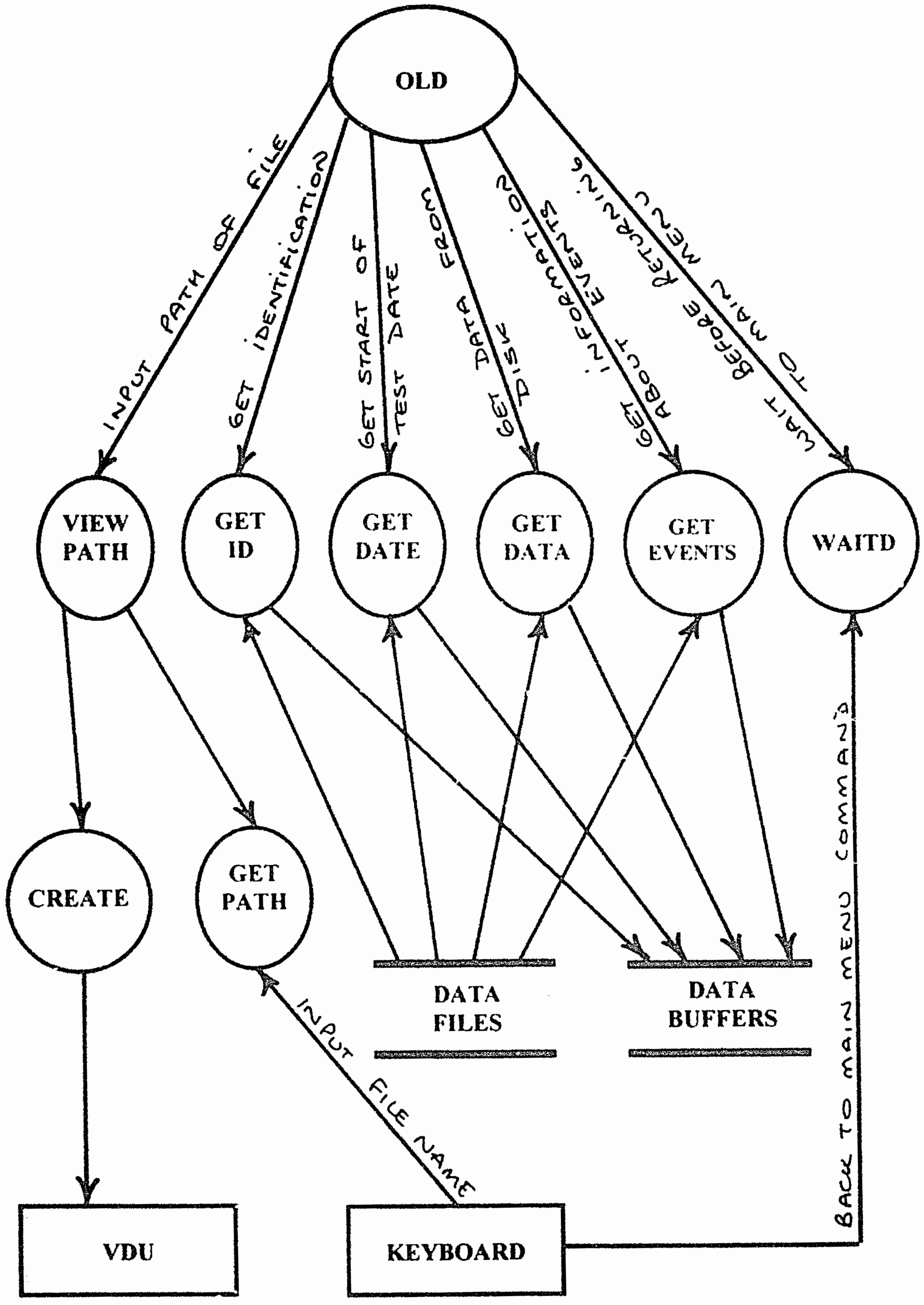


UPLOAD
DETECT START OF BLOCK
UPLOAD IDENT
UPLOAD START OF TEST DATE
UPLOAD NEW DATA
UPLOAD TIME OF EVENTS
*
IDENT
DATE
DATA
EVENTS
SERIAL PORT

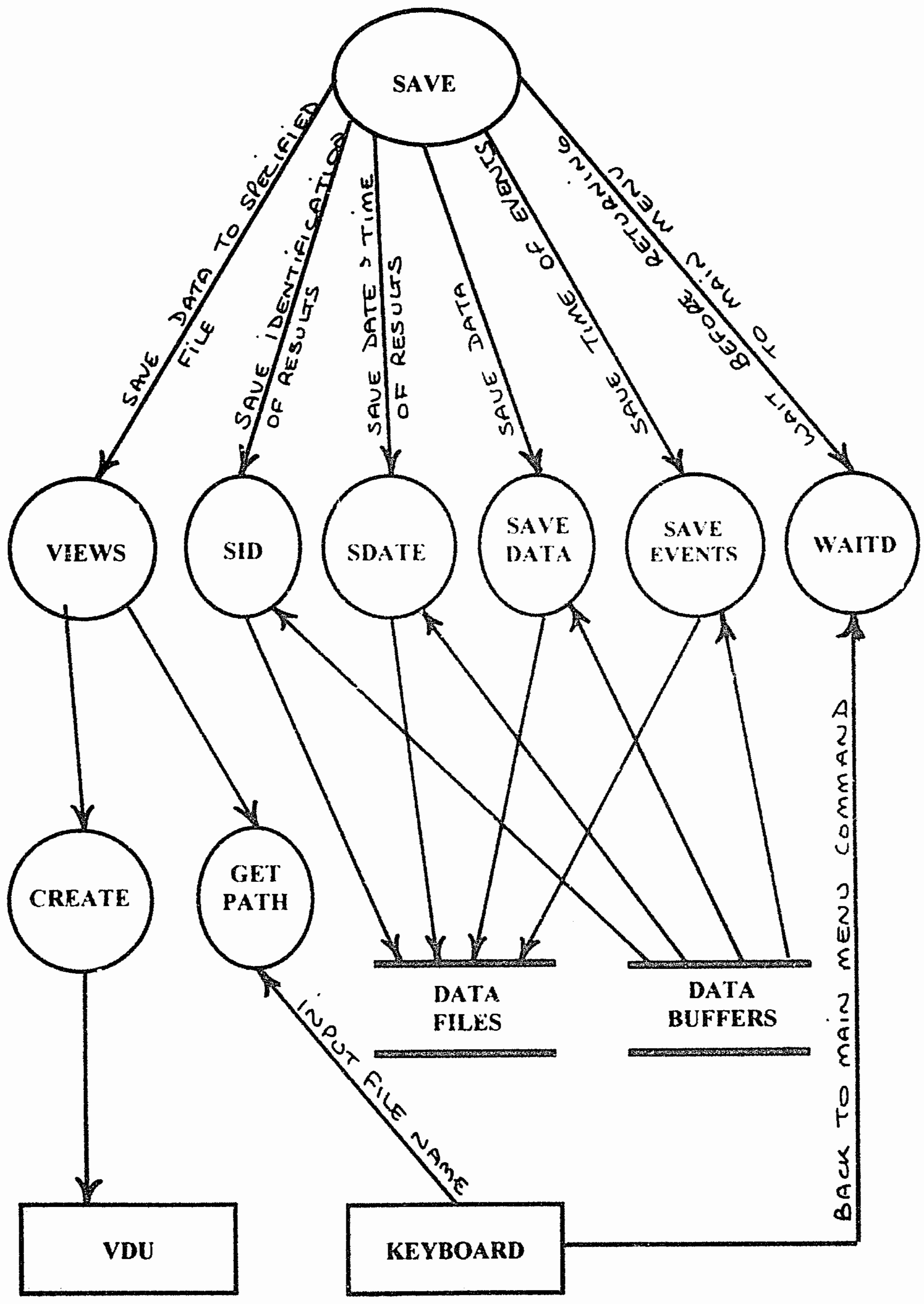
DFD LEVEL 3


DIR
INPUT NAME OF DIRECTORY
GET FIRST FILE
GET REST OF FILES
WAIT BEFORE RETURNING TO MAIN MENU
VIEW DIR
FIRST
MORE
WAITD
CREATE
FILE NAME
FILE NAMES
DATA FILES
CLEAR SCREEN
BACK TO MAIN MENU COMMAND
VDU
GET PATH
INPUT PATH
KEYBOARD

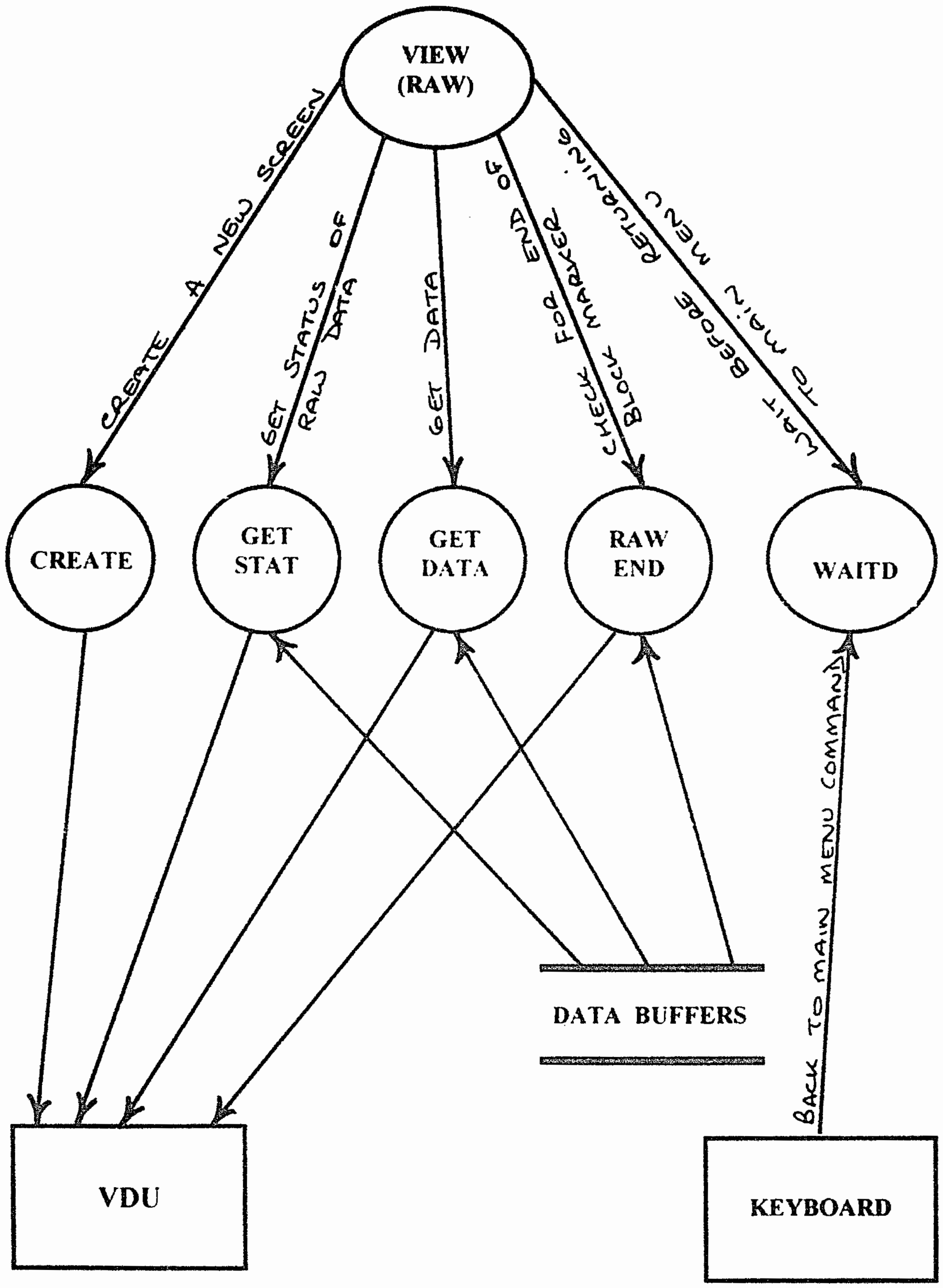
DFD LEVEL 3


OLD
INPUT PATH OF FILE
GET IDENTIFICATION
GET START OF TEST DATE
GET DATA FROM DISK
GET INFORMATION ABOUT EVENTS
WAIT BEFORE RETURNING TO MAIN MENU
VIEW PATH
GET ID
GET DATE
GET DATA
GET EVENTS
WAITD
CREATE
GET PATH
DATA FILES
DATA BUFFERS
INPUT FILE NAME
BACK TO MAIN MENU COMMAND
VDU
KEYBOARD

DFD LEVEL 3


SAVE
SAVE DATA TO SPECIFIED FILE
SAVE IDENTIFICATION OF RESULTS
SAVE DATE & TIME OF RESULTS
SAVE DATA
SAVE TIME OF EVENTS
WAIT BEFORE RETURNING TO MAIN MENU
VIEWS
SID
SDATE
SAVE DATA
SAVE EVENTS
WAITD
CREATE
GET PATH
DATA FILES
DATA BUFFERS
INPUT FILE NAME
BACK TO MAIN MENU COMMAND
VDU
KEYBOARD

DFD LEVEL 3

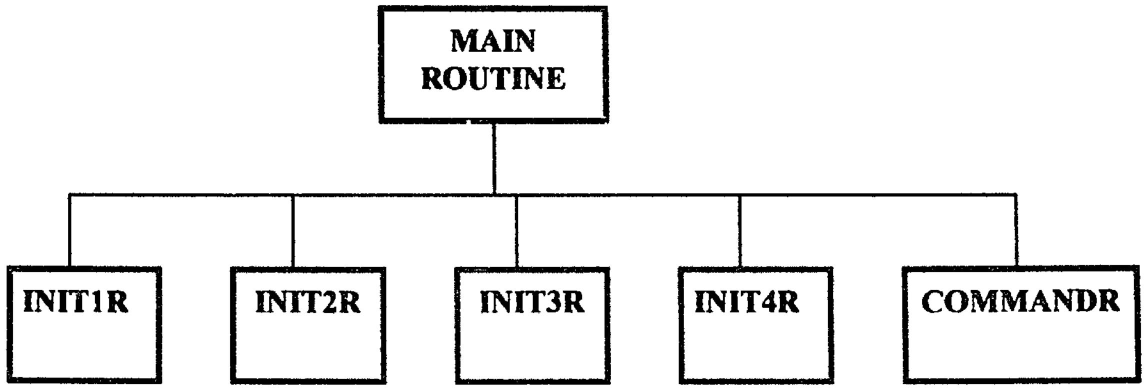

VIEW (RAW)
CREATE A NEW SCREEN
GET STATUS OF RAW DATA
GET DATA
CHECK FOR END OF BLOCK MARKER
WAIT BEFORE RETURNING TO MAIN MENU
CREATE
GET STAT
GET DATA
RAW END
WAITD
DATA BUFFERS
BACK TO MAIN MENU COMMAND
VDU
KEYBOARD

C.2.

The Data Processing System

Jackson structure diagrams

DPS PROGRAM STRUCTURE



1, 2, 3, 4

FUNCTIONS:

1. Initialise the serial port with parameters: 9600 baud rate, 8 data bits, 1 stop bit, and no parity checking.

2. Check the status of the serial port.

3. If the status of serial port is bad, then print an error message.

4. Return to the MAIN routine.

INIT2R ROUTINE

1, 2, 3, 4

FUNCTIONS:

1. Initialise the printer port.

2. Check the status of the printer port.

3. If the status of printer port is bad, then print an error message.

4. Return to the MAIN routine.

INIT3R ROUTINE

5, 6, 7

FUNCTIONS:

5. Set the screen mode to text mode 3.

6. DISPLAY routine (Print the MSGMENU to the VDU screen).

7. Return to the calling program.

INIT4R ROUTINE

1, 2, 3

FUNCTIONS:

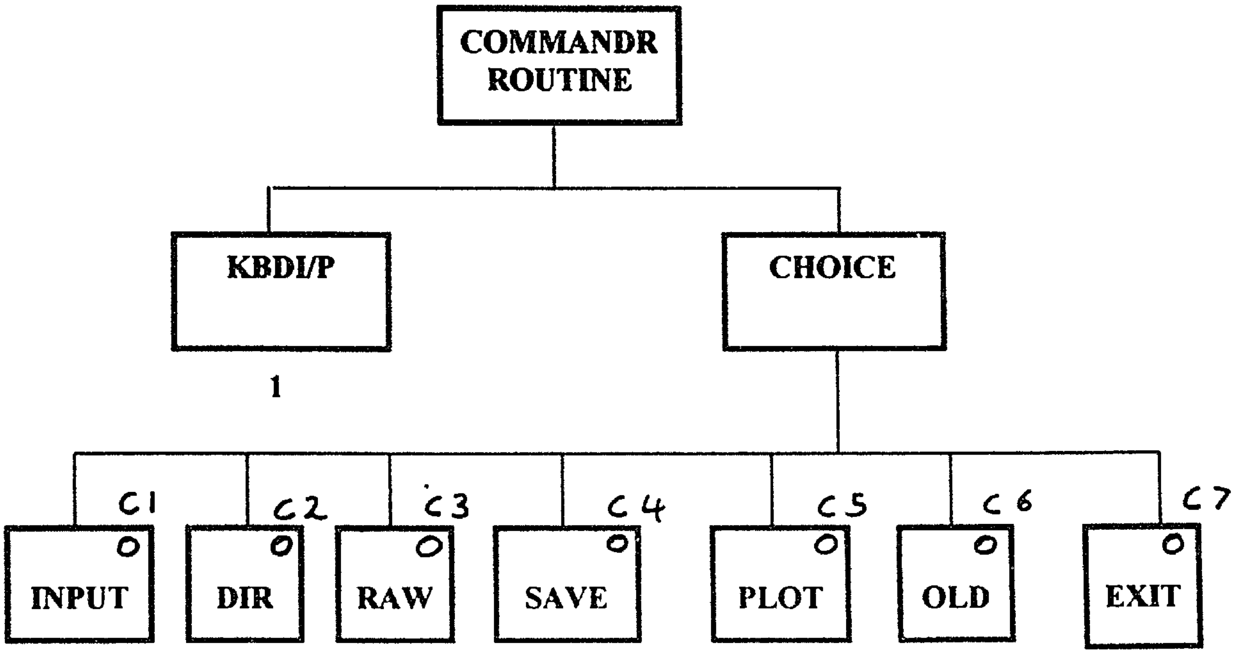
1. Initialise all the counters, pointers and variables.
2. Clear the data buffers.
3. Return to the MAIN routine.

DISPLAY ROUTINE

1

FUNCTIONS:

1. Print out the string of characters from, the address passed as a parameter upto the `$' character.

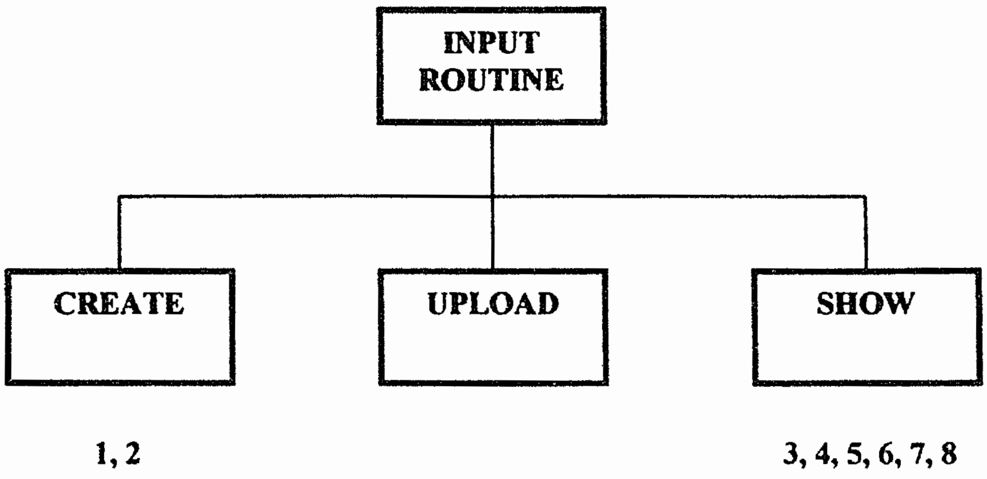


CONDITIONS:

C1.	Selected if `I'	key is pressed on the keyboard.
C2.	Selected if `D'	key is pressed on the keyboard.
C3.	Selected if `R'	key is pressed on the keyboard.
C4.	Selected if `S'	key is pressed on the keyboard.
C5.	Selected if `P'	key is pressed on the keyboard.
C6.	Selected if `O'	key is pressed on the keyboard.
C7.	Selected if `E'	key is pressed on the keyboard.

FUNCTIONS:

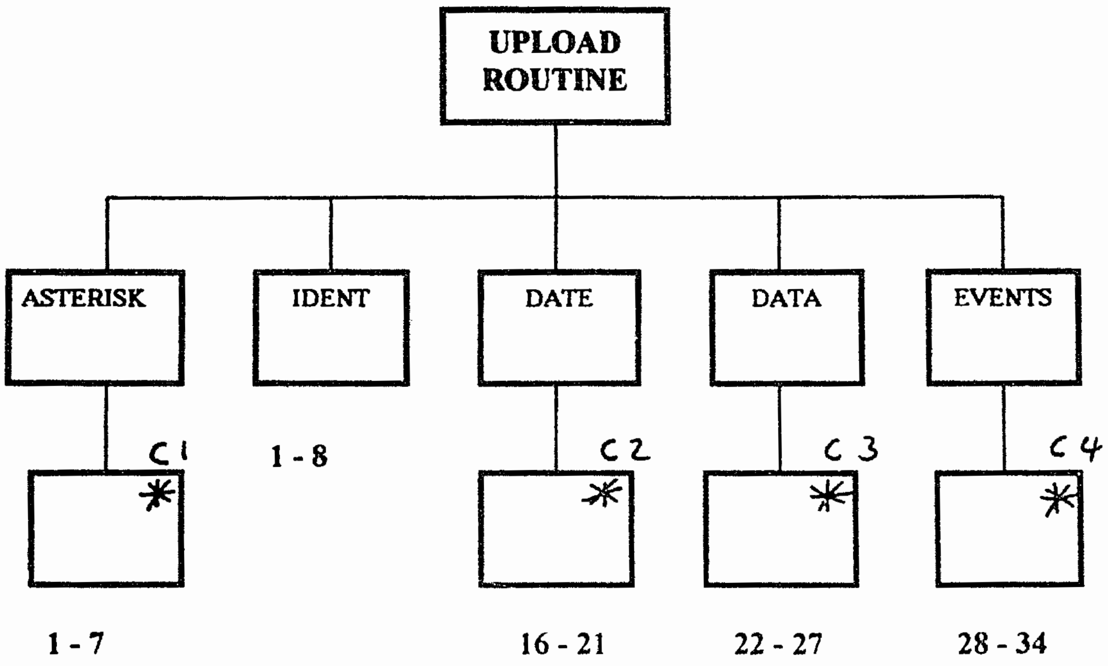
1.	Input a keystroke from the keyboard.



FUNCTIONS:

1. Set the VDU screen mode to text mode 3.
2. DISPLAY routine (Display MSGINPUT on the VDU screen).

3. Display the TMD identification value.
4. Display the 'start of test date'.
5. Display the number of data items received.
6. Display the number of events recorded.
7. Display 'Type 'C' to continue' on the VDU screen.
8. Return to the COMMANDR routine.



CONDITIONS:

C1. Repeat functions 23 - 29 until either:
an asterisk (*) is received, or
a KBD keystroke is detected.

C2. Receive until 12 serial port characters accepted.

C3. Input and store data until an '@' character is received.

C4. Input and store event times until an '#' character is received.

FUNCTIONS:

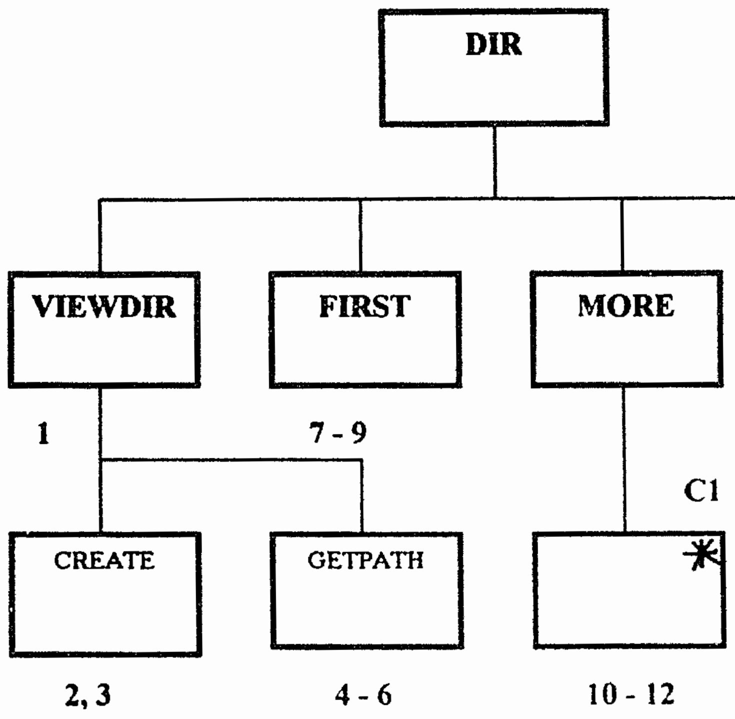
1. Receive one character from the serial port.
2. Test for a KBD character (escape mechanism).
3. If a keypress detected, then return to the INPUT routine with an error code.
4. Test the status of the serial port.
5. If the status is bad, then repeat functions 1 - 7.
6. Check the serial port character for an asterisk (*).
7. If no asterisk is detected, then repeat functions 1 - 7.

8. Receive one character from the serial port.
9. Test the status of the serial port.
10. If the status is bad, then repeat functions 8 - 10.
11. Store character in IDENTV.
12. Receive one character from the serial port.
13. Test the status of the serial port.
14. If the status is bad, then repeat functions 12 - 15.
15. Store character in IDENTV+1.

16. Set index register with address of DATEBUF
17. Receive one character from the serial port.
18. Test the status of the serial port.
19. If the status is bad, then repeat functions 17 - 19.
20. Store character in DATEBUF, increment index register.
21. Is the index register = DATEBUF + 12. No, repeat functions 17 - 21.

22. Set index register with address of BUFFER
23. Receive one character from the serial port.
24. Test the status of the serial port.
25. If the status is bad, then repeat functions 23 - 25.
26. Store character in DATEBUF, increment index register.
27. Is the character an '@' character. No, repeat functions 23 - 27.

28. Set index register with address of EVENTB
29. Receive one character from the serial port.
30. Test the status of the serial port.
31. If the status is bad, then repeat functions 29 - 31.
32. Store character in DATEBUF, increment index register.
33. Is the character an '#' character. No, repeat functions 29 - 33.
34. Return to the main menu control routine



CONDITIONS:

C1. Find and display matching filenames untill the last file is detected.

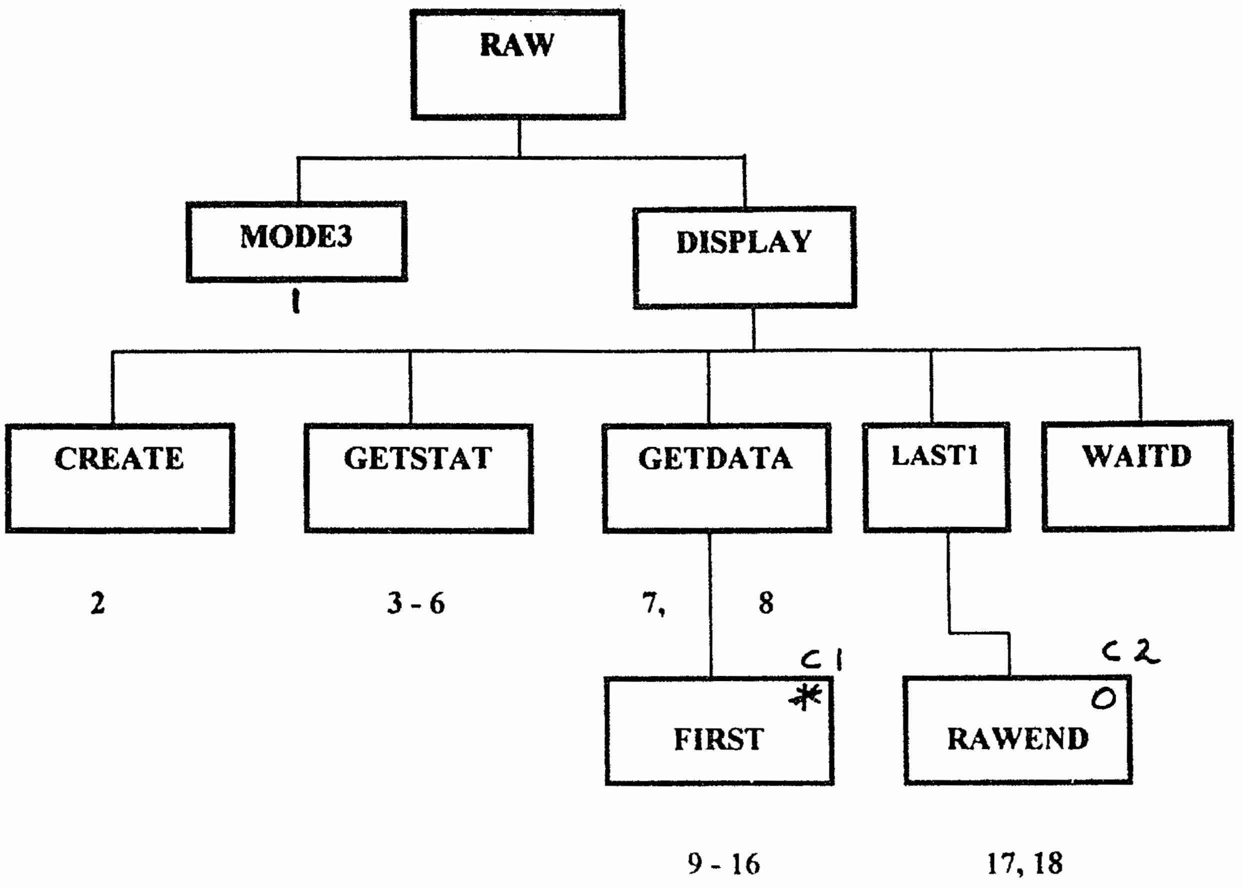
FUNCTIONS:

1. Move the pointer for MSGdir in the index register.
2. Set the VDU screen mode to text mode 3.
3. DISPLAY routine (Display MSGDI)R
4. Insert 64 (the size) into the PATH buffer.
5. Input from, the KBD, the path and file specification into PATH.
6. Insert an ASCIIZ character, into PATH, at the end of the file specification.

7. Set up a DTA.
8. Find the first matching directory entry.
9. Display the filename.

10. Move the cursor to a new screen position.
11. Fetch the next matching filename.
12. Display the filename.

13. Move the cursor to a new screen position.
14. Display MSG3, the return to the main menu routine.
15. Wait for a KBD keypress.
16. Call INIT4R to display MSGMENU.
17. Return to the main menu control routine



CONDITIONS:

C1.	Bytes of data read from the file untill an '@' character detected.
C2.	Selected if an '@' character is detected before a full screen of data has been read.

FUNCTIONS:

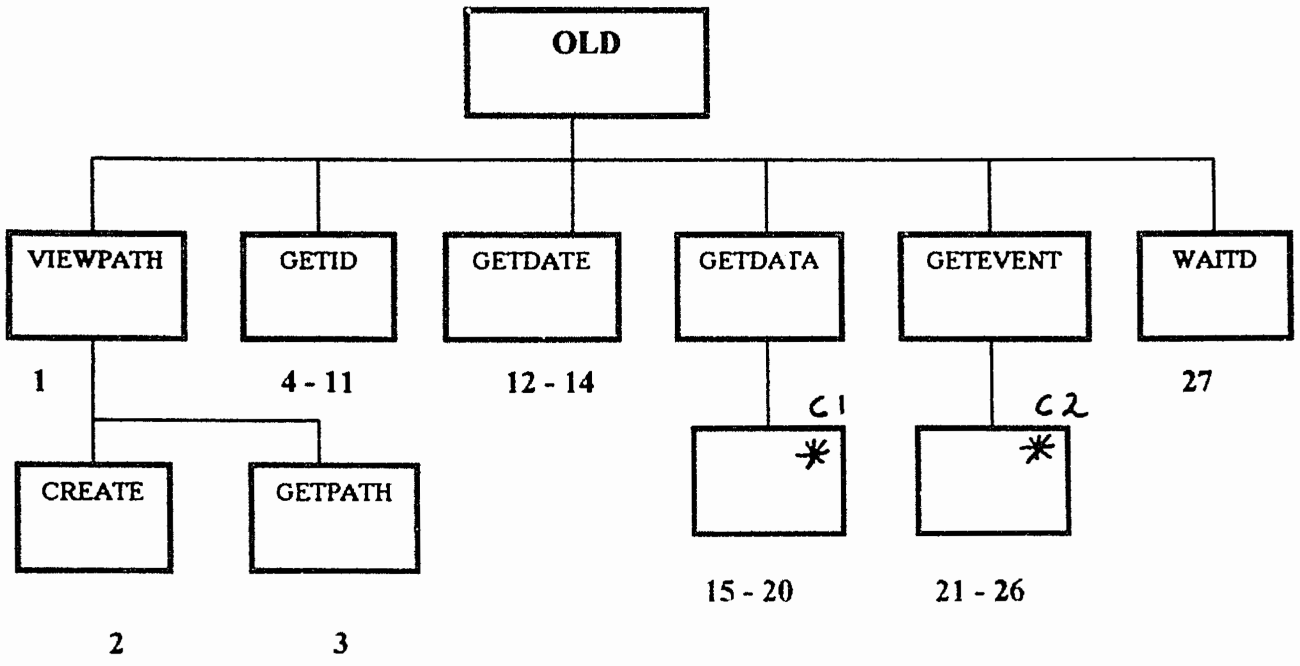
1. Set screen mode. move the pointer for MSGRAW into the index register.
2. DISPLAY ›utine (Create a new screen)

3. Move cursor.
4. Display TMD identification IDENT.
5. Move cursor.
6. Display the 'start of test' date DATE.

7. Move pointer for the start of BUFFER into an index register.
8. Move 16 into the loop counter register CX. (number of bytes of data per line).9

9. Move cursor.
10. Display 1 byte of data.
11. Increment the data buffer pointer.
12. Is this the last byte of data in the file? If 'yes', goto function 14.
13. Repeat functions 9 to 113.

14. Increment the row pointer ROW.
15. Is this the last row of the screen? If 'yes' exit loop.
16. Repeat functions 9 to 116.
17. Pop value from the CPU stack, correction because of exiting a loop mid-stream.
18. Return to the main control routine COMMANDR.



CONDITIONS:

C1.	Read bytes of data untill an '@' character is detected.
C2.	Read bytes of data untill an '#' character is detected.

FUNCTIONS:

1.	Mov the pointer value for MSGLOAD into the index register
2.	Create a new screen.
3.	Input the filename and directory path. Store in PATH.
4.	Open the specified file.
5.	Jump to function 81 if a loading error is detected.
6.	Read 1 byte from the opened file.
7.	Jump to function 81 if a loading error is detected.
8.	Display the character
9.	Load the TMD identification (2 bytes) and store in IDENT.
10.	Return to the OLD routine.
11.	Display "FILE LOADING ERROR", then return to OLD routine.

FUNCTIONS: continued

12. Load next 12 bytes into DATEBUF.
13. Jump to function 81 if a loading error is detected.
14. Return to OLD routine.

15. Read 1 byte from opened file.
16. Jump to function 81 if a loading error is detected.
17. check for an '@' character.
18. Repeat functions 85 to 88 if the '@' is not detected.
19. Display the '@' character.
20. Return to OLD routine.

21. Read 1 byte from opened file.
22. Jump to function 81 if a loading error is detected.
23. check for an '#' character.
24. Repeat functions 85 to 88 if the '#' is not detected.
25. Display the '#' character.
26. Return to OLD routine.

27. Call the WAITD routine to display the main menu MSGMENU and then return to the main control routine COMMANDR.

$\boxed{\textbf{EXIT}}$

1, 2

FUNCTIONS:

1. write one character 'E' to the screen

2. exit back to MSDOS

SDATA

1 - 6

FUNCTIONS:

1. Establish how many bytes of data are in BUFFER.
2. Check for an `@' character.
3. If 4094 bytes are counted before the `@' character is detected goto function 122.
4. Write bytes of data to the opened file.
5. If a file transfer error is detected goto function 122.
6. Return to the SAVE routine.

SEVENT

1 - 6

FUNCTIONS:

1. Establish how many bytes of data are in EVENTB.
2. Check for an `#' character.
3. If 254 bytes are counted before the `#' character is detected goto function 122.
4. Write bytes of data to the opened file.
5. If a file transfer error is detected goto function 122.
6. Return to the SAVE routine.

SID

1 - 9

FUNCTIONS:

1. Set up a DTA.
2. Create a new file using PATH.
3. Store handle in HANDLE.
4. Move an '*' into the DTA
5. Move the contents of IDENTV into the DTA.
6. Write 3 bytes into the opened file.
7. Return to the SAVE routine.

8. In the case of a file transfer error display MSG4
9. Return to the SAVE routine.

SDATE

1 - 4

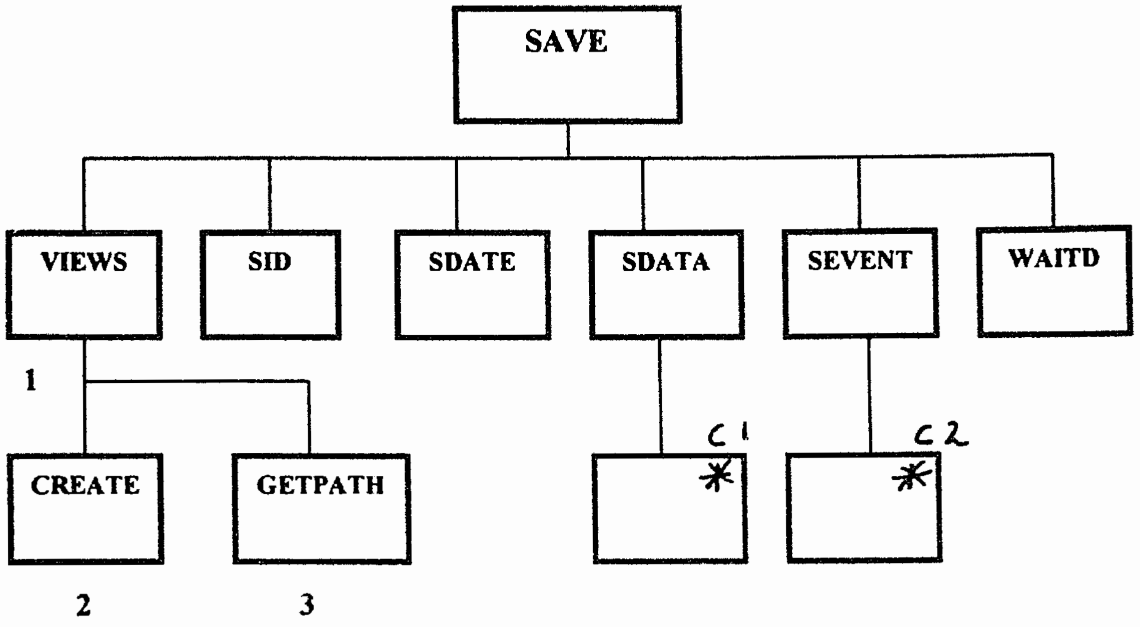
FUNCTIONS:

1. Move the contents of DATEBUF into the DTA.

2. Write 12 bytes to the opened file.

3. If a file transfer error is detected execute functions: 122 and 123.

4. Return to the SAVE routine.



CONDITIONS:

C1. Save bytes of data from BUFFER untill an '@' character is detected.

C2. Save words of event times from EVENTB untill a '#' character is detected.

FUNCTIONS:

1. Move the pointer for MSGSAVE into the index register.

2. Create a new screen.

3. Input a new file specification into the PATH variable.

C.3.

The Data Processing System

program listing

```
;##################################################################
;####                DPSPROG.ASM                             ####
;####                                                        ####
;####                9 - 07 - 1993                           ####
;####                                                        ####
;####                Mike  Wetton                            ####
;####                                                        ####
;##################################################################

_TEXT       SEGMENT

        ASSUME      CS:_TEXT , DS:_TEXT , SS:_TEXT
        ORG         100h

START:      JMP     MAIN


******************************************************************
        PROGRAM'S DATA AREA OF MEMORY FOR ITS VARIABLES
******************************************************************

ERRCODE     DB      0

HANDLE      DW      0                               ; a handle to an opened file

PATH        DB      64    DUP (00h)                 ; file specification

DTA         DB      64    DUP (00h)                 ; Data Transfer Area

IDENTV      DB      5Ah , 0a5h                      ; TMD identification

DATEBUF DB    31h, 33h, 30h, 37h, 39h, 33h, 30h, 39h, 34h, 35h, 30h, 30h
                                                    ; date and time values

ROW         DB      1                               ; row on screen
COL         DB      1                               ; column on screen
```

```
**********************************************************************
        PROGRAM DATA AREA OF MEMORY FOR VDU SCREEN
                        MESSAGES
**********************************************************************

MSG1 DB     "Serial port not initialised." , 0Dh , 0Ah , "$"

MSG2 DB     "Printer port not initialised." , 0Dh , 0Ah , "$"

MSG3 DB     "Type `C' to return to the main MENU screen.","$"

MSG4 DB     "  **** LOADING FILE CONTENTS ERROR ****",
0Dh,0Ah , "$"

MSGMENU
        DB     "                                  ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `E'   to  EXIT  program; back to DOS.     ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `I'   to upload data from the serial port.",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `D'   to view a directory of files.       ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `R'   to view the raw date from memory.   ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `L'   to load the raw data from a file.   ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `S'   to save the raw data to  a file     ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "  Type   `P'   to view the processed data.         ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "                                  ",0dh,0ah
        DB     "$"
```

```
MSGINPUT
        DB    "                                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "  UPLOADING  RAW  DATA  FROM  THE TMD or the DTD
",0dh,0ah
        DB    "  ***********************************************
",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "  IDENTIFICATION  OF  THE TMD   =              ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "  STARTING  DATE  OF TEST       =             ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "$"



MSGRAW
        DB    "                                  ",0dh,0ah
        DB    "  DISPLAYING  THE  RAW  DATA  FROM  THE
MEMORY",0dh,0ah
        DB    "  ********************************************
",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "  IDENTIFICATION  OF  THE  TMD   =             ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "  STARTING  DATE  OF  TEST                    ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "$"

MSGDIR
        DB    "                                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "     DISPLAYING  A  DIRECTORY  OF  FILES
",0dh,0ah
        DB    "     ***********************************         ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "   Input the complete path of the directory.       ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "   For example,    A:\test\*.dat                ",0dh,0ah
        DB    "      or        C:*.*                  ",0dh,0ah
        DB    "                                  ",0dh,0ah
        DB    "$"
```

```
MSGLOAD
        DB    "                                   ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "    LOADING DATA FROM A SPECIFIED FILE
",0dh,0ah
        DB    "    ****************************************
",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "   Input the complete path and filename.          ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "   For example,   A:\test\test.dat             ",0dh,0ah
        DB    "      or         C:trial.dat               ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "$"


MSGSAVE
        DB    "                                   ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "      SAVING DATA TO A SPECIFIED FILE
",0dh,0ah
        DB    "     ***************************************
",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "   Input the complete path and filename          ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "   For example,   A:\test\test3.dat             ",0dh,0ah
        DB    "      or         C:trial56.dat               ",0dh,0ah
        DB    "                                   ",0dh,0ah
        DB    "$"


******************************************************************
PROGRAM'S DATA  AREA OF MEMORY FOR DATA AND EVENTS
******************************************************************

BUFFER      DB    4096   DUP (00h)

EVENTB      DB     256   DUP (00h)
```

```
***************************************************************************
                      THE MAIN CONTROL MODULE
***************************************************************************

MAIN: MOV  AX , CS
      MOV  DS , AX
      MOV  ES , AX
      NOP
      CALL INIT1R                 ; initialise serial port
      NOP
      CALL INIT2R                 ; initialise printer
      NOP
      CALL INIT3R                 ; initialise the VDU screen
      NOP
      CALL COMMANDR               ; control routine
      NOP
      MOV  AH , 0
      INT  16h
      NOP
      INT  3                      , exit to DEBUG
      NOP
      MOV  AH , 4Ch               . DOS exit function
      MOV  AL , 00
      INT  21h

MESSAGE    PROC NEAR

      MOV  AH , 02h               . move cursor
      MOV  BH , 00h
      INT  10h
      NOP
      MOV  AH , 09h               . write message to screen
      MOV  DX , SI
      INT  21h
      NOP
      RET

MESSAGE    ENDP
```

```
INIT1R        PROC NEAR

        MOV  AH , 0                     ; ititialise serial port
        MOV  DX , 0
        MOV  BX , 0
        MOV  AL , 0E3h       ; 9600 baud,8-data bits,1 stop bit & no parity
        INT  14h
        NOP
        MOV  AH , 03                    ; get serial port status
        INT  14h
        NOP
        MOV  AH , 01                    ; send one character to the serial
port
        MOV  DX , 0
        MOV  BX , 0
        MOV  AL , 0Dh                   ; character to be sent
        INT  14h
        NOP
        RET

 INIT1R ENDP


INIT2R        PROC

        MOV  AH , 01                    ; initialise printer
        MOV  DX , 0                     ; 0 - LPT1
        INT  17h
        NOP
        MOV  AH , 02                    ; get printer status
        MOV  DX , 0                     ; 0   LPT1
        INT  17h
        NOP
        MOV  AH , 0                     ; send a character to the printer
        MOV  DX , 0                     ; 0 - LPT1
        MOV  AL , 0Dh                   ; character to be printed
        INT  17h
        NOP
        RET

INIT2R        ENDP
```

```
INIT3R      PROC NEAR

        MOV  AH , 00h                      ; set screen mode
        MOV  AL , 03h
        INT  10h
        NOP
        MOV  SI , offset MSGMENU
        CALL DISPLAY
        NOP
        RET

INIT4R      ENDP


*******************************************************************
        COMMANDR CONTROLLING MENU FUNCTIONS MODULE
*******************************************************************

COMMANDR        PROC NEAR

KBDINP:     MOV  AH , 00h                      ; input form keyboard
        INT  16h
        NOP
CHOICE: CMP        AL , 49h
        JE   INPUTR
        CMP  AL , 69h
        JNE  DIRC
INPUTR: CALL       INPUT                   ; call serial data input routine
        JMP  NEXT
        NOP
DIRC: CMP  AL , 44h
        JE   DIRR
        CMP  AL , 64h
        JNE  RAWC
DIRR: CALL  DIR                            ; call routine to view a directory of
files
        JMP  NEXT
        NOP
RAWC:       CMP  AL , 52h
        JE   RAWR
        CMP  AL , 72h
        JNE  SAVEC
RAWR:       CALL  RAW                      ; call routine to view raw data
        JMP  NEXT
        NOP
```

```
SAVEC:      CMP   AL , 53h
       JE     SAVER
       CMP    AL , 73h
       JNE    PLOTC
SAVER:      CALL  SAVE                 ; call routine to save the raw data
       JMP    NEXT
       NOP
PLOTC:      CMP   AL , 50h
       JE     PLOTR
       CMP    AL , 70h
       JNE    OLDC
PLOTR:      CALL  PLOT                 ; call routine TO print/plot results
       JMP    NEXT
       NOP
OLDC:CMP    AL , 4Ch
       JE     OLDR
       CMP    AL , 6Ch
       JNE    EXITC
OLDR:CALL   OLD                        ; call routine to load raw data from
a file
       JMP    NEXT
       NOP
EXITC:      CMP   AL , 45h
       JE     EXITR
       CMP    AL , 65h
       JNE    NEXT
EXITR:      CALL  EXIT                 ; call routine to enable an exit back
to DOS
       JMP    NEXT
       NOP
       RET

COMMANDR          ENDP


DISPLAY PROC      NEAR

       MOV    AH , 09h                 ; display A string of characters
       MOV    DX , SI
       INT    21h
       RET

DISPLAY     ENDP
```

```
*********************************************************************
      UPLOAD TMD INFORMATION VIA THE SERIAL PORT
*********************************************************************

INPUT       PROC NEAR

      MOV  SI , offset MSGINPUT
      NOP
      CALL CREATE                 ; routine to create a new screen
      NOP
      CALL UPLOAD                 ; routine to upload the serial data
      NOP
      CALL SHOW                 ; routine to show the identification of
the TMD
      NOP
      RET

INPUT       ENDP

CREATE      PROC NEAR

      MOV  AH , 00h               ; set screen mode
      MOV  AL , 03h
      INT  10h
      NOP
      CALL DISPLAY
      NOP
      RET

CREATE      ENDP

UPLOAD      PROC NEAR

      MOV  ERRCODE , 00
ASTERISK: NOP
      MOV  AH , 02                ; move cursor
      MOV  DX , 0
      MOV  BX , 0
      INT  14h
      NOP
      MOV  CX , AX
      NOP
      MOV  AH , 01                ; detect a KBD keystroke
      INT  16h
      JE   EEEE
      JMP  BACKER
      NOP
```

```
EEEE:TEST  CH , 80h
     JNE   ASTERISK
     NOP
     CMP   CL , 2Ah                    ; start of block marker
     JNE   ASTERISK
     JMP   IDENT
     NOP
IDENT:     MOV  AH , 02                ; move cursr
     MOV   DX , 0
     MOV   BX , 0
     INT   14h
     NOP
     MOV   CX , AX
     NOP
     TEST  CH , 80h
     JNE   ASTERISK
     NOP
     MOV   IDENTV , CL
     NOP
AAAA:      MOV  AH , 02                ; move cursor
     MOV   DX , 0
     MOV   BX , 0
     INT   14h
     NOP
     MOV   CX , AX
     NOP
     TEST  CH , 80h
     JNE   AAAA
     NOP
     MOV   IDENTV+1 , CL
     NOP
DATE:MOV   DI , offset DATEBUF         ; date and time buffer
     NOP
BBBB:MOV   AH , 02                     ; move cursor
     MOV   DX , 0
     MOV   BX , 0
     INT   14h
     NOP
     MOV   CX , AX
     NOP
     TEST  CH , 80h
     JNE   BBBB
     NOP
     MOV   [DI] , CL
     INC   DI
     CMP   DI , offset DATEBUF+12
     JNE   BBBB
     NOP
```

```
DATA:           MOV   DI , offset BUFFER    ; data buffer
        NOP
CCCC:MOV   AH , 02                          ; move cursor
        MOV   DX , 0
        MOV   BX , 0
        INT   14h
        NOP
        MOV   CX , AX
        NOP
        TEST  CH , 80h
        JNE   CCCC
        NOP
        MOV   [DI] , CL
        INC   DI
        CMP   CL , 40h                      ; end of data marker
        JNE   CCCC
        NOP
EVENT:          MOV   DI , offset EVENTB    ; event times buffer
        NOP
DDDD:           MOV   AH , 02               ; move cursor
        MOV   DX , 0
        MOV   BX , 0
        INT   14h
        NOP
        MOV   CX , AX
        NOP
        TEST  CH , 80h
        JNE   DDDD
        NOP
        MOV   [DI] , CL
        INC   DI
        CMP   CL , 24h                      ; end of block marker
        JNE   DDDD
        NOP
        RET

BACKER: MOV         ERRCODE , 0FFh
        NOP
        RET

UPLOAD          ENDP
```

```
SHOWPROC NEAR

        MOV   ROW , 07h          ; display the TMD identification
        MOV   COL , 26h
        CALL  CURSOR
        MOV   SI , offset IDENTV
        CALL  CONVERT
        CALL  SEECHAR
        INC   SI
        CALL  CONVERT
        CALL  SEECHAR
        NOP
        MOV   ROW , 09h          ; display the TMD 'start of test' date
        MOV   COL , 26h
        CALL  CURSOR
        MOV   SI , offset DATEBUF
        MOV   CX , 3
FFFF:   PUSH  CX
        MOV   DH , [SI]
        INC   SI
        MOV   DL , [SI]
        INC   SI
        CALL  SEECHAR
        NOP
        MOV   DH , 20h
        MOV   DL , 20h
        CALL  SEECHAR
        NOP
        POP   CX
        LOOP  FFFF
        NOP
        MOV   ROW , 12h
        MOV   COL , 01h
        CALL  CURSOR
        MOV   SI , offset MSG3             ; display return to menu message
        CALL  DISPLAY
        NOP
        MOV   AH , 0             ; wait for a KBD keypress
        INT   16h
        NOP
        CALL  INIT4R
        RET

SHOWENDP
```

```
****************************************************************
GENERAL ROUTINES USED BY ANY MAJOR FUNCTIONAL
OUTINES
****************************************************************

CURSOR       PROC NEAR

        MOV  AH , 02                    ; move the cursor to a specified
position
        MOV  BX , 0
        MOV  DH , ROW
        MOV  DL , COL
        INT  10h
        RET

CURSOR       ENDP


CONVERT PROC    NEAR

        MOV  DH , [SI]                  ; convert top 4 bits to ASCII
        MOV  CL , 4
        SHR  DH , CL
        AND  DH , 0Fh
        OR   DH , 30h
        CMP  DH , 39h
        JBE  MISS1
        ADD  DH , 07h
        NOP
MISS1:       MOV  DL , [SI]             ; convert bottom 4 bits to ASCII
        AND  DL , 0Fh
        OR   DL , 30h
        CMP  DL , 39h
        JBE  MISS2
        ADD  DL , 07h
        NOP
MISS2:       RET

CONVERT ENDP
```

```
SEECHAR PROC     NEAR

        MOV  AH , 0Eh                      ; write 1 character to the screen
(high bits)
        MOV  BX , 0
        MOV  AL , DH
        INT   10h
        MOV  AH , 0Eh                      ; write 1 character to the screen
(low bits)
        MOV  BX , 0
        MOV  AL , DL
        INT   10h
        NOP
        RET

SEECHAR ENDP

*******************************************************************
        DISPLAY THE CONTENTS OF A SPECIFIED DIRECTORY
*******************************************************************


DIR     PROC NEAR

        CALL  VIEWDIR               ; routine to create a screen and input a path
        NOP
        CALL  FIRST                 ; routine to display the first directory entry
        NOP
        CALL  MORE                  ; routine to display directory entries
        NOP
        CALL  WAITD                 ; display return to main MENU routine
        NOP
        RET

DIR     ENDP

VIEWDIR PROC     NEAR

        MOV  SI , offset MSGDIR
        NOP
        CALL  CREATE                ; display MSGDIR
        NOP
        CALL  GETPATH               ; input path and files pecification
        NOP
        RET

VIEWDIR ENDP
```

```
GETPATH PROC      NEAR

          MOV  SI , offset PATH               ; insert the size of PATH buffer
          MOV  BYTE PTR[SI] , 64
          MOV  AH , 0Ah                       ; input ASCII string (path)
          MOV  DX , offset PATH
          INT  21h
          NOP
          MOV  DI , offset PATH               ; insert an ASCIIZ character
          MOV  BH , 0
          MOV  BL , PATH+1
          ADD  BL , 2
          ADD  DI , BX
          MOV  BYTE PTR[DI] , 0
          NOP
          RET

GETPATH ENDP

FIRST PROC NEAR

          MOV  AH , 1Ah                       ; set up the DTA
          MOV  DX , offset DTA
          INT  21h
          NOP
          MOV  AH , 4Eh                       ; find first matching directory entry
          MOV  CX , 0
          MOV  DX , offset PATH+2
          INT 21h
          NOP
          MOV  SI , offset DTA                ; display file name
          ADD  SI , 1Eh
          MOV  ROW , 0Eh
          MOV  COL , 0Ch
          CALL CURSOR
          NOP
NEXTC:         MOV  AL , [SI]
          CMP  AL , 0
          JE   EXITFN
          NOP
          NOP
          MOV  AH , 0Eh                       ; display character
          MOV  BX , 0
          INT  10h
          NOP
          INC  SI
          JMP  NEXTC
EXITFN: RET
FIRST ENDP
FIRST ENDP

MOREPROC NEAR

NEXTO:      NOP
        INC     ROW                         ; new line
        CMP     ROW , 18h
        JB      MISSADD
        MOV     ROW , 0Eh
        ADD     COL , 12h
        NOP
MISSADD: CALL       CURSOR
        NOP
        MOV     SI , offset DTA             ; address of new filename
        ADD     SI , 1Eh
        NOP
        MOV     AH , 4Fh                    ; fetch next matching filename
        INT     21h
        JB      EXITMN
        NOP
NEXTI:      MOV   AL , [SI]
        CMP     AL , 0
        JE      BACK2
        NOP
        NOP
        MOV     AH , 0Eh                    ; display character
        MOV     BX , 0
        INT     10h
        NOP
        INC     SI
        JMP     NEXTI
BACK2:      JMP   NEXTO
        NOP
EXITMN: RET

MOREENDP

WAITD       PROC NEAR

        MOV     ROW , 18h
        MOV     COL , 14h
        CALL    CURSOR
        MOV     SI , offset MSG3        ; display return to menu message
        CALL    DISPLAY
        NOP
        MOV     AH , 0                      ; wait for a KBD keypress
        INT     16h
        NOP
        CALL    INIT4R
        NOP
        RET
```

```
WAITD      ENDP
*******************************************************************
           LOAD TMD TEST DATA FROM A SPECIFIED FILE
*******************************************************************

RAW  PROC NEAR

      MOV  SI , offset MSGRAW
      NOP
      CALL  CREATE
      NOP
GETSTART:  MOV  ROW , 04h              ; display the TMD identification
      MOV  COL , 26h
      CALL  CURSOR
      MOV  SI , offset IDENTV
      CALL  CONVERT
      CALL  SEECHAR
      INC   SI
      CALL  CONVERT
      CALL  SEECHAR
      NOP
      MOV  ROW , 06h                   ; display the TMD 'start of test' date
      MOV  COL , 26h
      CALL  CURSOR
      MOV  SI , offset DATEBUF
      MOV  CX , 3
GETDATA:   PUSH  CX
      MOV  DH , [SI]
      INC   SI
      MOV  DL , [SI]
      INC   SI
      CALL  SEECHAR
      NOP
      MOV  DH , 20h
      MOV  DL , 20h
      CALL  SEECHAR
      NOP
      POP   CX
      LOOP  HHHH
      NOP
IIII:      MOV  ROW , 09h              ; display the TMD data
      MOV  COL , 00h
      MOV  SI , offset BUFFER
      MOV  CX , 16
JJJJ:      PUSH  CX
      NOP
      CALL  CURSOR
LAST1:  MOV          AL , [SI]
      CMP  AL , 40h
```

```
        JE    RAWEND
        CALL  CONVERT
        CALL  SEECHAR
        INC   SI
        ADD   COL , 04
        NOP
        POP   CX
        LOOP  JJJJ
        MOV   CX , 16
        MOV   COL , 0
        INC   ROW
        CMP   ROW , 24
        JNE   JJJJ
        NOP
BACKR:        CALL  WAITD
        NOP
        RET

RAWEND: POP CX
        JMP   BACKR

RAW   ENDP



*******************************************************************
            SAVE TMD TEST DATA TO A SPECIFIED FILE
*******************************************************************

SAVE PROC NEAR

        CALL  VIEWS              ; display instructions, get file specifications
        NOP
        CALL  SID                      ; save the TMD identification
        NOP
        CALL  SDATE                    ; save the date and time values
        NOP
        CALL  SDATA                    ; save the TMD data
        NOP
        CALL  SEVENT                   ; save the recorded event times
        NOP
        CALL  WAITD                    ; wait for a user keypress before
                                         returning to COMMANDR
        NOP
        RET

SAVE ENDP
```

```
VIEWS       PROC NEAR

       MOV  SI , offset MSGSAVE
       NOP
       CALL CREATE                        ; create a new screen
       NOP
       CALL GETPATH                       ; get the file specification
       NOP
       RET
VIEWS       ENDP

SID    PROC NEAR

       MOV  AH , 1Ah                      ; set up a DTA
       MOV  DX , offset DTA
       INT  21h
       NOP
       JB   SAVE_ERR
       NOP
       MOV  AH , 3Ch                      ; create a new file
       MOV  DX , offset PATH+2
       MOV  CX , 0
       INT  21h
       NOP
       JB   SAVE_ERR
       MOV  HANDLE , AX
       NOP
       MOV  DI , offset DTA
       MOV  BYTE PTR[DI] , 2Ah
       NOP
       MOV  AL , IDENTV
       MOV  [DI + 1] , AL
       NOP
       MOV  AL , IDENTV+1
       MOV  [DI +2 ] , AL
       NOP
       MOV  AH , 40h                      ; write 3 bytes to file
       MOV  BX , HANDLE
       MOV  CX , 3
       MOV  DX , DI
       INT  21h
       JB   SAVE_ERR
       NOP
       RET
SAVE_ERR: MOV     SI , offset MSG4        ; display an error message
       CALL CREATE
       NOP
       RET
```

```
SID    ENDP



SDATE       PROC NEAR

        MOV  SI , offset DATEBUF
        MOV  DI , offset DTA
        MOV  CX , 12
        REP  MOVSB
        NOP
        MOV  AH , 40h                    ; write 12 bytes to file
        MOV  BX , HANDLE
        MOV  CX , 12
        MOV  DX , offset DTA
        INT  21h
        NOP
        JB   SAVE_ERR
        NOP
        RET
SDATE       ENDP

SDATA       PROC NEAR

        MOV  SI , offset BUFFER
        MOV  CX , 0
NEXTBY: MOV      AL , [SI]
        INC  SI
        INC  CX
        CMP  CX , 4095
        JE   SAVE_ERR
        CMP  AL , 40h
        JNE  NEXTBY
        NOP
        MOV  AH , 40h                    ; write CX bytes to file
        MOV  BX , HANDLE
        MOV  DX , offset BUFFER
        INT  21h
        NOP
        JB   SAVE_ERR
        NOP
        RET

SDATA       ENDP
```

```
SEVENT      PROC NEAR

        MOV   SI , offset EVENTB
        MOV   CX , 0
NEXTBZ: MOV       AL , [SI]
        INC   SI
        INC   CX
        CMP   CX , 254
        JE    SAVE_ERR
        CMP   AL , 23h
        JNE   NEXTBZ
        NOP
        MOV   AH , 40h                   ; write CX bytes to file
        MOV   BX , HANDLE
        MOV   DX , offset EVENTB
        INT   21h
        NOP
        JB    SAVE_ERR
        NOP
        MOV   AH , 3Eh                   ; close file
        MOV   BX , HANDLE
        INT   21h
        NOP
        RET

SEVENT      ENDP


*****************************************************************
              PLOT THE TMD RESULTS ON A VDU SCREEN OR ON
                              A PRINTOUT
*****************************************************************

PLOT  PROC  NEAR

        MOV   AH , 0Eh                    ;  write one character for testing
program
        MOV   AL , 50h
        MOV   BX , 0
        INT   10h
        NOP
        RET

PLOT  ENDP
```

```
OLD   PROC NEAR

        CALL  VIEWPATH                ; get file specification
        NOP
        CALL  GETID                   ; get TMD identification
        NOP
        CALL  GETDATE                 ; get date and time values
        NOP
        CALL  GETDATA                 ; get the TMD data
        NOP
        CALL  GETEVENT                ; get the times of recorded events
        NOP
        CALL  WAITD                   ; wait for a user KBD keypress
        NOP
        RET

OLD   ENDP


VIEWPATH PROC    NEAR

        MOV   SI , offset MSGLOAD
        NOP
        CALL  CREATE                  . create a new screen routine
        NOP
        CALL  GETPATH                 . input a filename and path
        NOP
        RET

VIEWPATH ENDP


GETIDPROC NEAR

        MOV   AH . 3Dh                . open the specified file
        MOV   AL , 0                  ; read only
        MOV   DX . offset PATH·2
        INT   21h
        NOP
        JB    LOADERR                 ; to detect a file error
        MOV   BX , AX                 ; transfer the file handle
        MOV   HANDLE , AX
        NOP
        MOV   AH . 3Fh                ; read 1 byte of file
        MOV   CX . 1
        MOV   DX , offset DTA
        INT   21h
        NOP
        MOV  ROW , 18
        MOV  COL , 0
        CALL CURSOR
        NOP
        MOV  AH , 0Eh                   ; display character
        MOV  AL , DTA
        MOV  BX , 0
        INT  10h
        NOP
        JB   LOADERR
        NOP
        MOV  AH , 3Fh                   ; read 2 bytes of file
        MOV  BX , HANDLE
        MOV  CX , 2
        MOV  DX , offset IDENTV
        INT  21h
        NOP
        JB   LOADERR
        NOP
        RET

LOADERR:
        MOV  SI , offset MSG4           ; display an error message
        CALL CREATE
        NOP
        RET

GETIDENDP


GETDATE PROC     NEAR

        MOV  AH , 3Fh                   ; read 12 bytes of file
        MOV  CX , 12
        MOV  DX , offset DATEBUF
        INT  21h
        NOP
        JB   LOADERR
        NOP
        RET

GETDATE ENDP


GETDATA PROC     NEAR

        MOV  DX , offset BUFFER
LOOPD:       MOV  AH , 3Fh              ; read 1 byte of file
        MOV  BX , HANDLE
        MOV  CX , 1
        INT   21h
        NOP
        JB    LOADERR
        NOP
        MOV  SI , DX
        INC   DX
        MOV  AL , [SI]
        CMP  AL , 40h
        JNE   LOOPD
        NOP
        MOV  AH , 0Eh                    ; display character
        MOV  BX , 0
        INT   10h
        NOP
        RET

GETDATA ENDP


GETEVENT PROC    NEAR

        MOV  DX , offset EVENTB
ELOOP:        MOV  AH , 3Fh              ; read 1 byte of file
        MOV  BX , HANDLE
        MOV  CX , 1
        INT   21h
        NOP
        JB    LOADERR
        NOP
        MOV  SI , DX
        INC   DX
        MOV  AL , [SI]
        CMP  AL , 23h
        JNE   ELOOP
        NOP
        MOV  AH , 0Eh                    ; display character
        MOV  BX , 0
        INT   10h
        NOP
        MOV  AH , 3Eh                    ; close the file
        MOV  BX , HANDLE
        INT   21h
        NOP
        JB    LOADERR
        NOP
        RET

GETEVENT ENDP
```

APPENDIX D

The Software Development Environment Details

This section contains:

D.1.

Environment Description

Environment Description

Programs are required to edit the source program and for the serial communication between the PC and the Motorola evaluation board. The names of the main programs used for this project are as follows:

M	Microsoft's full-screen editor,
PASM	Motorola's portable asembler,
UBUILDS	to create S-records,
MSKERMIT	for serial communications, and
BUFFALO	to accept S-records and commands to debug a user program.

The four PC programs used for software development were packaged into an efficient environment by calling them from within an MS-DOS batch file (written by Mike Wetton). The batch file invokes two macine code programs. One that clears the screen and selects forground and background colours for text. The other allows the user to select menu choices from within a batch file. A TYPE command inside the batch file creates a menu on the screen (see diagram below). The PC screen would show the following menu:

```
Type `1'   for     EDITING
Type `2'   for     PASM
Type `3'   for a   LISTING
Type `4'   for     S-RECORDS
Type `5'   for     MSKERMIT
Type `6'   for     MS-DOS
```

D.2.

A menu batch file

MC68HC11 ASSEMBLY LANGUAGE PROGRAMMING ENVIRONMENT

(author Mike Wetton, file MENU.BAT)

```
ECHO  OFF
BREAK ON
COLOUR

:START

CLS
TYPE  MESSAGE.TXT
:INKEY
ERROR
IF ERRORLEVEL 6 GOTO DOS
IF ERRORLEVEL 5 GOTO COMMUNICATE
IF ERRORLEVEL 4 GOTO UBUILDS
IF ERRORLEVEL 3 GOTO LIST
IF ERRORLEVEL 2 GOTO PASM
IF ERRORLEVEL 1 GOTO EDIT
GOTO INKEY

:EDIT

CD  EDIT
M  A:\PROGS\NEW.ASM
CD..
GOTO  START


:PASM

CD  PASM
PASMHC11 -dxs -l A:\PROGS\NEW.LST A:\PROGS\NEW.ASM
CD..
ECHO  Type '1' to continue
ERROR
GOTO  START
```

```
:LIST

TYPE  A:\PROGS\NEW.LST
ECHO  Type `1' to continue
ERROR
GOTO  START


:UBUILDS

CD  PASM
UBUILDS NEW.O
COPY   NEW.MX A:\PROGS\NEW.MX
ECHO   Type `1' to continue
ERROR
CD..
GOTO  START

:COMMUNICATE

CD  KERMIT
MSKERMIT
ECHO  Type `L' to download SRECORDS
ECHO  Type `1' to continue
ERROR
CD..
GOTO  START

:END

START
:DOS
```

D.3.

An assembly language choice program

THE MENU CHOICE PROGRAM (author Mike Wetton, file COLOUR.COM)

The assembly language program COLOUR.COM is invoked from the menu batch file (MENU.BAT). The instructions of the COLOUR.COM program select the foreground and background colours for the screen menu.

```
START:    MOV  AH , 00          ; select text screen mode
          MOV  AL , 03
          INT  10h
          NOP
          MOV  AX , 0B800h      ; CGA screen memory
          MOV  ES , AX
          MOV  DI , 0000        ; start of screen memory
          MOV  CX , 07D0h       ; 2000 bytes of screen memory
          MOV  AL , 20h         ; space character
          MOV  AH , IFh         ; attribute:-  white on blue
          REP  STOSW            ; write 2000 blue spaces
          NOP
          MOV  AH , 4Ch         . return to DOS function
          MOV  AL, 00
          INT  21h
```

INPUT MENU CHOICE PROGRAM (author Mike Wetton, file ERROR.COM)

The assembly language program ERROR.COM is invoked from the menu batch file
(MENU.COM) in order to select a choice of menu options ASCII codes 31h to 36h are input from the KBD then converted to BCD values 1 to 6 and returned to DOS as an error code.

```
START:    MOV  AH , 00          ; wait for a KBD input
          INT  16h
          NOP
          CMP  AL , 31h         ; less than '1' ?
          JB   START
          CMP  AL , 36          ; greater than '6' ?
          JB   START
          NOP
          MOV  AH , 4Ch         ; return to DOS function
          SUB  AL , 30h         ; error code for DOS use
          INT  21h
```

APPENDIX E

The Hardware Development Environment Details

This section contains:

THE MC68HC11EVBU EVALUATION BOARD

THE MC68HC11EVB EVALUATION BOARD

THE MC68HC11EVM EVALUATION BOARD

THE MC68HC11EVBU EVALUATION BOARD

This board is designed to test and debug systems that use the MC68HC11 in it's single-chip mode. The MC68HC11E9 microcontroller has on-chip RAM, EEPROM and a ROM containing a monitor program called BUFFALO.

The EVBU contains two support chips; a real-time clock/calendar chip with 32 bytes of static RAM, and a serial interface chip used to communicate with the terminal I/O port.

The terminal I/O baud rate defaults to 9600 baud and communcation is via the microcontroller SCI subsystem.

The EVBU requires a, user-supplied, +5 volt dc power supply and a RS232C compatible terminal for operation.

THE MC68HC11EVB EVALUATION BOARD

This EVB evaluation board was designed to demonstrate the capabilities of the MC68HC11 microcontroller unit. It operates in either the debugging mode or the evaluation mode.

The debugging mode allows the user to debug user code under the control of the BUFFALO monitor program. The user code can be assembled on the host computer (the IBM PC) and downloaded as S-records into the EVB RAM, or assembled one-line at a time by the EVB assembler/disassembler.

This board is designed to expand the memory of the microcontroller and provide protected interfaces to the user's target circuitry.

The EVB and EVM have two serial links to a development system, namely, the ***host port*** and the ***terminal port.*** The host port is used for downloading user programs, whereas, the terminal port is used to monitor program execution.

RS232C line drivers and receivers are used as a serial interface between the EVB and the host computer and terminal.

The EVB has a fixed 9600 baud rate provided for the host port, and a hardware selectable (300 - 9600) baud rate for the terminal port.

The EVBU requires a, user-supplied, +5 V, +12V, -12V and GND dc power supply and a RS232C compatible terminal for operation.

THE MC68HC11EVM EVALUATION BOARD

This board was designed to allow the user to program the MC68HC11 series of microcontrollers in the single-chip or the expanded multiplexed mode of operation. The Motorola literature states that it is a tool for designing, debugging and evaluating the MC68HC11 microcontroller based target system equipment.

The EVM is the most sophisticated of all the aforementioned evaluation boards, as it contais pseudo ROM and EEPROM memory

The user has the choice of interfacing directly to the MC68HC11 ports, or to the ports via buffered I/O gates.

The EVM also requires a, user-supplied, +5 V, +12V, -12V and GND dc power supply and a RS232C compatible terminal for operation.