UNIVERSAL MEASUREMENT SYSTEM CONTROLLER

by

İLHAN NEDİM ALP

B.S. in E.E., TECHNICAL UNIVERSITY OF İSTANBUL, 1982

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Boğaziçi University

1986

UNIVERSAL MEASUREMENT SYSTEM

CONTROLLER

APPROVED BY

Y. Doc. Dr. Ahmet Denker.....

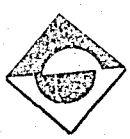
Doc. Dr. Omer Cerid.....

Doc. Dr. Yekta Ulgen.....

DATE OF APPROVAL :....21./X./1986.....

ACKNOWLEDGEMENTS

I consider it a privilege to acknowledge those people who helped and encouraged me during my graduate education and specially my thesis. First, I would like to thank Dr. Ahmet Denker, my thesis advisor for his kind help. I would like to thank Dr. Yorgo Istefanopulos, my graduate advisor, I also would like to thank Mr. Tanju Argun and Dr. Hayri Mutluay who let me use the facilities of the research and development laboratories of NETAS.



UNIVERSAL MEASUREMENT SYSTEM CONTROLLER

This thesis describes the design and implementation of a measurement instrumentation system controller by using IEEE 488 standard bus (IEEE 488 is also called GPIB General Purpose Interface Bus, within the thesis we will use GPIB instead of IEEE 488) named as UMSC (Universal Measurement System Controller).

GPIB is a widely used digital communication standard in instrumentation. The most famous companies manufacture their instruments with GPIB. They can be interconnected by using GPIB, in order to realize the complex test sets. At that point, a controller is needed to control the communication between them. However the controllers mentioned above are very expensive devices. If you need one or more small test sets with remote control, using them is not feasible economically. So dedicated small controllers are necessary. The aim of this work is to satisfy this requirement.

The UMSC is designed as a microprocessor-based system. It utilizes the GPIB talker, listener and controller functions. It is a programmable device, to realize special test algorithms. It provides a line editor to edit user programs, and a compiler to check the syntax of the user programs and to interpret them. It also offers a basic language developed for this work to write user programs.

GENEL AMAÇLI ÖLÇME DÜZENEK YÖNETGEÇİ

Bu tez çalışması IEEE 488 standart veri yolu kullanan bir ölçme aletleri dizgesi yönetgeçinin tasarımını ve gerçeklenmesini tanımlar. (IEEE 488, GPIB "General Purpose Interface Bus" olarakta adlandırlır. Tezin içeriğinde IEEE 488 yerine GPIB kullanılacaktır).

GPIB, ölçme aletlerinde yaygın olarak kullanılan bir sayısal iletişim standandartıdır. Tanınmış birçok firma ölçme aletlerini GPIB özelliği ile üretmektedir. Bu aletler, karmaşık test düzeneklerini gerçeklemek için, GPIB üzerinden birbirlerine bağlanabilirler. Bu noktadan bakıldığında ölçme aletlerinin birbirleriyle iletişimini yöneteçek bir yönetgeçe gereksinim duyulur. Oysaki yukarıda sözü geçen yönetgeçler çok pahalı aletlerdir. Eğer bir veya birden fazla küçük test düzeneklerine gereksinim varsa, adı geçen yönetgeçler parasal açıdan uygun olmazlar. Bu neden ile özel olarak yapılmış küçük yönetgeçler gereklidir. Bu çalişmanin amaçi böyle bir isteğe yanit vermektir.

UMSC mikro islemçili bir dizge olark tasarlanmiştir. GPIB konuşmaçi (talker), dinleyici (listener) ve yönetgeç (controller) işlemlerini gerçekler. Özel test algoritmalarını gerçekleştirebilmek için programlanabilen bir aletdir. Kullanıcı programlarını yazabilmek için bir program yazıcısı ve bu programların yazılımını derleyen, makine diline çeviren bir derleyiciyi sağlar. Ayni zamanda bu çalişma için geliştirilen

bir programlama dilinide sağlar.

TABLE OF CONTENTS

I. INTRODUCTION

In the design phase of such devices, the first question that would confront the designer should be "What kind of people will use this device ?". If a specialist will use it, the device can be designed with relatively less emphasis upon man-machine interface. But if an average person will use it, the design should be more informative than the first one. The answer to this question shapes the main structure of the system. Although generally these kinds of devices are used by specialists, the more informative system is desired from the point of user friendliness.

This kind of system have to provide

... a CRT monitor to follow comments results and program parts easily

... an understandable language for interfacing system and user.

... an editor to write user programs

... a syntax checker for the written user programs

... an interpreter to execute the user programs

... a flexible hardware for future developments, such as increasing memory capacity

In such systems, the user programs are stored in disks,diskettes or magnetic tapes. However this is an expensive way to a small controller. Therefore a nonvolatile type of memory is preferably chosen for storing user programs.

The UMSC was designed with above properties. It also offers the GPIB controller, talker, and listener functions as its main aim.

UMSC works with the stored program logic. It consists of two parts: System Software and System Hardware. System block diagram is given in Fig.1.1

The microprocessor is the main control element in the system. It executes all required logical functions, and controls I/O devices.

The UMSC contains 7 I/O units. Two of them are serial ports, the other two are parallel ports. There is a timer. The last two ports are GPIB unit ports. One of the serial I/O devices is used for communication with an interactive CRT terminal and the other one is used for communication with a printer. These two serial I/O devices use the RS232C Standard Serial Communication Interface circuit. Parallel I/O circuit could be used for communication with any other microcomputer in future, if necessary, or it could be used to control a relay circuit or any other test sets The last two I/O ports are used to communicate with the microprocessor and GPIB intreface circuit.

The memory circuit consists of three different types of memory chips: EPROM circuit for system S/W, EEPROM for user S/W, and RAM circuit for data and stack area.

The UMSC S/W consists of two main modules. Those are are background and foreground modules.

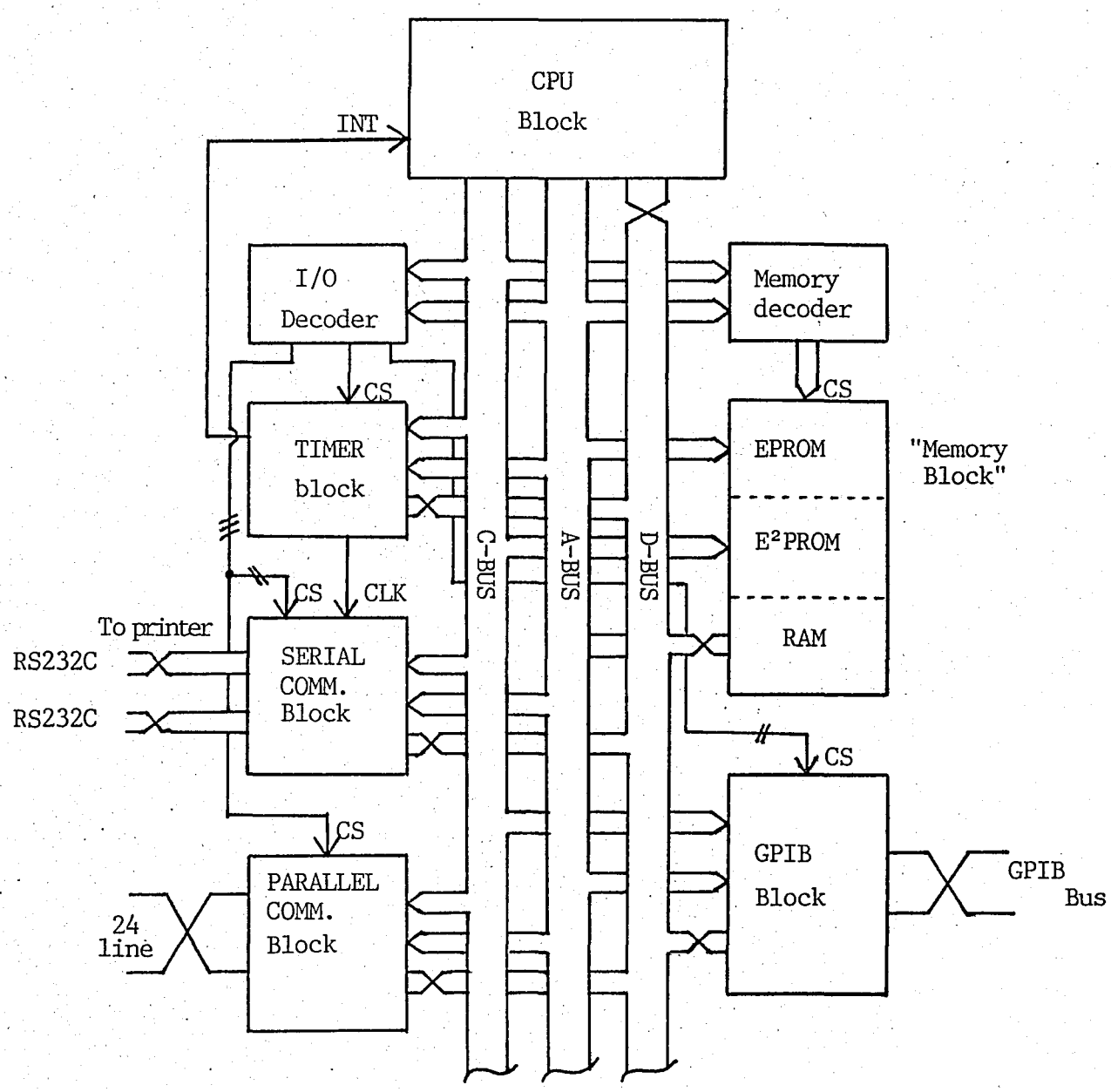


Fig.1.1 System block diagram of UMSC

In chapter 2, the concept of GPIB is described. The communication methods in GPIB, description of controller, talker and listener functions as well as electrical and mechanical aspects are defined briefly.

Chapter 3 gives the concept of message used in GPIB. The message structure is recommended by IEEE 728 Standard.

In chapter 4, the hardware description of UMSC is given in full detail. It also describes the memory and I/O port organisation, based upon the software structure of system.

In chapter 5, software structure of UMSC is explained. The S/W structure has a state driven architecture. Some important routines are described and illustrated by flow diagrams. This chapter is supported by source listings (see appendix).

Chapter 6 is written as a user manual for UMSC. It explains the editor, list, compiler and monitor modes with examples. It also describes UMSC language. It finishes with the describtion of error messages.

II. WHAT IS THE IEEE 488 (GPIB) ?

A. INTRODUCTION

The experience of designing systems for a variety of applications in the early 1970's caused Hewlett-Packard to define a standard intercommunication mechanism which would allow them to easily assemble instrumentation systems of varying degrees of complexity. For a particular application each instrument designer designed his/her own interface from scratch. But none was consistent in terms of electrical levels, pin-outs on a connecter, and types of connectors. Every time they built a system they had to createnew cables and new documentation just to specify the cabling and interconnection procedures.

Based on this experience, Hewlett-Packard began to define a new interconnection scheme. In 1972, Hewlett-Packard came out with the first version of the bus which since has been modified and standardized by a committee of several manufacturers, coordinated through the IEEE, to perfect what is now known as the IEEE 488 Interface Bus (also known as the HP-IB, the GPIB and the IEC bus). While this bus specification may not be perfect, it is a good compromise of the various desires and goals of instrumentation and computer peripheral manufacturers to produce a commun interconnection mechanism. It fits most instrumentation systems in use today and also fits very well the microcomputer I/O bus requirements. The basic design objectives for the GPIB were to:

1.Specify a system that is easy to use, but has all of the terminology and the definitions related to that system preciesly spelled out so that everyone uses the same language when discussing the GPIB.

2.Define all of mechanical, and electrical interface requirements of a system, yet not define any of the aspects (they are left up to the instrument designer).

3.Permit a wide range of capabilities of instruments and computer peripherals to use a system simultaneously and not degrade each other's performance.

4.Allow different manufacturers' equipment to be connected together on the same bus.

5.Define a system that is good for limited distance interconnections.

6.Define a system with minimum restrictions on the performance of the devices.

7.Define a bus that allows asynchronous communication with a wide range for the data rates.

8.Define a low cost system that does not require extensive and elaborate interface logic for the low cost instruments, yet provides higher capability for the higher cost instruments if desired.

9.Allow systems to exist that do not need a central controller; that is, communication directly from one instrument to another is possible.

Although the GPIB was originally designed for

instrumentation systems, it became obvious that most of these systems would be controlled by a calculator or computer. With this in mind several modifications were made to the original proposal before its final adoption as an international standard. The major characteristics of the GPIB are listed below:

Data Rate:

1M bytes/s. max.

250k bytes/s. typ.

Multiple Devices:

15 devices, max. (electrical limit)

8 devices, typ. (interrupt flexibility)

Bus Length:

20 m, max.

2 m, typ.

Byte-Oriented:

8-bit commands

8-bit data

Block Multiplexed:

Optimum strategy on GPIB due to setup overhead for commands

Interrupt Driven:

Serial poll(slower devices)

Parallel poll(faster devices)

Direct Memory Access:

One DMA facility at controller serves all devices on bus

Asynchronous:

One talker

Multiple listeners

I/O to I/O Transfers:

Talker and listeners need not to include a microcomputer/controller

The bus can be best understood by examining each of these characteristics.

1. DATA RATE

Certainly, the 250k bytes/s data rate can be easily achived by the IEEE 488 bus is sufficient for microcomputers and their peripherals, and is more than needed for typical analog instruments that take only a few readings per second. The 1M-byte/s maximum data rate is not easily achieved on the GPIB and requires special attention.

2. MULTIPLE DEVICES

With the GPIB, up to 8 devices can be handled easily by a controller; with some slowdown in interrupt handling, up to 15 devices can work together. The limit of 8 is imposed by the number of unique parallel poll responses available; the limit of 15 is set by the electrical drive characteristics of the bus. Logically, the IEEE 488 Standard is capable of accommodating more device addresses (31 primary, each potentially with 31 secondaries).

3. BUS LENGTH

The GPIB is designed typically to have 2m of length per device, and totaly 20m. This is enough for most instrumentation systems.

4. BYTE ORIENTED

The GPIB has 8-bit wide data path that may be used to transfer ASCII or binary data as well as the necessary status and control bytes.

5. BLOCK MULTIPLEXED

The GPIB is, by nature a block multiplexed bus due to the overhead involved in addressing various devices to talk and listen. This overhead is less bothersome if it only occurs once for a large number of data bytes (once per block).

6. INTERRUPT DRIVEN

The GPIB has two interrupt protocols. The first is a single service request line that may be asserted by all interrrupting devices. The controller then polls all devices to find out which wants service. The polling mechanism is well defined and can be easily automated. For higher performance, the parallel poll capability in the IEEE 488 allows up to 8 devices to be polled at once -each device is assigned to one bit of the data bus. This mechanism provides fast recognition of an

interrupting device. A drawback is the frequent need for the controller to explictly conduct a parallel poll, since there is no equivalent of the service request line for this mode.

7. DIRECT MEMORY ACCESS (DMA)

In many applications, no immediate processing of I/O data on a byte-by-byte basis is needed or wanted. With the GPIB, one DMA facility at the controller serves all devices.

8. ASYNCHRONOUS TRANSFER

An asynchronous bus is desirable so that each device can transfer at its own rate. The GPIB is asynchronous and uses a special 3-wire handshake that allows data transfers from one talker to many listeners.

9. I/O TO I/O TRANSFERS

In practice, I/O to I/O transfers are seldom done due to the need for processing data and changing formats or due to mismatched data rates. However, the GPIB can support this mode of operation where the microcomputer is neither the talker nor one of the listeners.

B. GPIB SIGNAL LINES

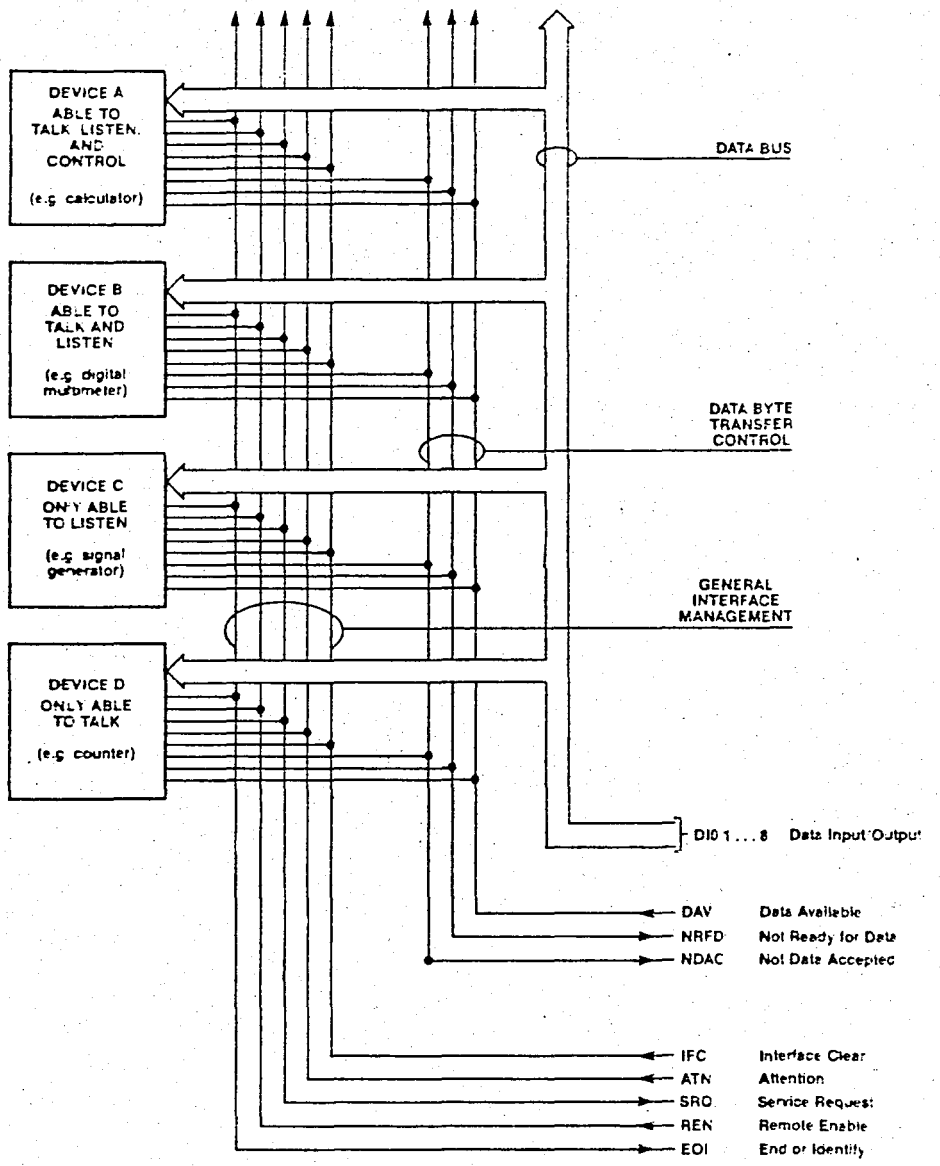


Fig.2.1 GPIB Lines and Device Configuration

1. DATA BUS

The data bus is an 8-bit bidirectional bus.The lines DIO1 through DIO8 are used to transfer addresses, control information

and data. The formats for addresses and control bytes are defined by the IEEE 488 Standard. Data formats are undefined and may be ASCII (with or without parity) or binary. DI01 is the least significant bit (note that this will correspond to bit 0 on most computers).

The information transferred on the BUS in BIT PARALLEL BYTE SERIAL form. The transfer of the 3 byte sequence "BUS" is illustrated in Fig.2.2

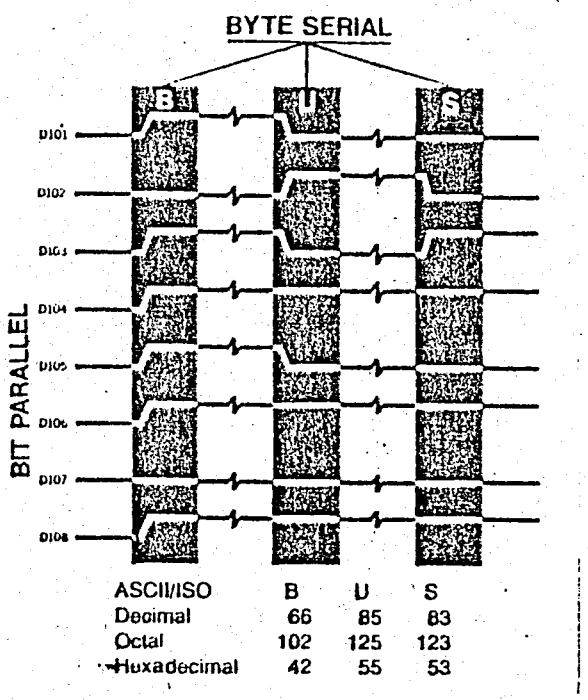


Fig.2.2 Data bus format.

2. GENERAL INTERFACE MANAGEMENT BUS

It consists of 5 lines. They are used to manage an orderly flow of information across the interface

a. ATTENTION (ATN)

This signal is asserted by the Controller to indicate that it is placing an address or control byte on the Data Bus. ATN is de-asserted to allow the assigned Talker to place status or data on the Data Bus. The Controller regains control by re-asserting ATN; this is normally done synchronously with the handshake to avoid confusion between control and data bytes.

When true ATN (asserted) places the interface in the COMMAND MODE where all devices accept data on the interface and interpret it as COMMANDS or ADDRESSES. When false ATN (de-asserted) places the interface in the DATA MODE where the active talker sources device dependent DATA to all active listeners.

COMMAND MODE (ATN true): The commands serve several different purposes:

1.Talk or listen addresses select the instruments that will source and accept data. They are all multiline messages (i.e., messages sent over the data bus). Addresses are sent to all devices.

2.Universal commands cause every instrument so equipped to perform a specific interface operation. They include five multiline commands and four uniline commands.

3.Addressed commands are similar to universal commands, except that they affect only those devices that are addressed and are all multiline commands. An instrument responds to an addressed command, however, only after a controller has already

told it to be a listener.

4. Secondary commands are multiline messages that are always used in conjuction with an address, universal command, or addressed command, (also referred to as primary commands) to provide additional command codes. Thus they extend the code space when necessary.

TALK and LISTEN ADDRESSES: Every GPIB device has at least one device address. They are used by the active controller in the COMMAND MODE to specify who talks (via a Talk Address) and who listens (via Listen addresses). Any given Device Address can specify two correnspondings address codes on the Data Lines (although it may only actually respond to one):

1. Talk Address
2. Listen Address

The sixth and seventh buts (DIO6-DIO7) are used to distinguish between a device's talk and listen addres characters. Two address codes are used to tell every device to UNTALK or UNLISTEN.

UNIVERSAL COMMANDS: Universal commands are consist of two types, those are Multiline commands and Uniline commands.

MULTILINE COMMANDS: Device Clear Command (DCL): The universal device clear command causes all recognizing devices to return to a pre-defined device-dependent state. Recognizing devices respond whether they are addressed or in remote only. Device manuals define the reset state for each device recognizes the command.

Local Lockout Command (LLO): The local lockout command disables a particular front-panel or rear-panel local-reset or return-to-local (push button) on devices that recognize the command. Recognizing devices accept the command whether they are addressed or in remote only. REN must be set false to re-enable the push button, this also replaces all devices under local control.

Serial Poll Enable Command (SPE): The serial poll enable command establishes serial poll mode for all responding talker devices on the bus. When they are addressed to talk, each responding device will return a single eight-bit byte of status from each device. Devices which recognize this command must have Talker interface capabilities to allow the device to output the status-byte.

Serial Poll Disable Command (SPD): The serial poll disable command terminates serial poll mode for all responding devices to their normal state where they output device-dependent data rather than status information.

Parallel Poll Unconfigure Command (PPU): The parallel poll unconfigure command resets all parallel poll devices to the idle state (unable to respond to a parallel poll).

Untalk Command (UNT): The untalk command unaddresses the current talker. Sending an unused talk address would accomplish the same thing. This command is provided for convenience since addressing one talker automatically unaddress others.

Unlisten Command (UNL): The unlisten command unaddresses

all current listeners on the bus. Single listeners cannot be unaddressed without unaddressing all listeners. It is necessary that this command be used to guarantee that only desired listeners are addressed.

UNILINE COMMANDS:

Interface Clear (IFC)

Remote Enable (REN)

Attention (ATN)

Identify (IDY) : EOI+ATN

ADDRESSED COMMANDS:

Group Execute Trigger Command (GET): The group execute trigger command causes all devices which have the GET capability and are currently addressed to listen to initiate a preprogrammed action (e.g., trigger, take a sweep, etc.). Some devices may also recognize a device-dependent data character or string for this function (equivalent but requires entry into DATA MODE). The GET command provides a means of triggering devices simultaneously.

Selected Device Clear Command (SDC): The selected device clear command resets device currently addressed to listen to a device-dependent state (e.g. turn-on state, open all relays, etc.). Device manuals define the reset state for each device that recognizes the command. Same as DCL.

Go to Local Command (GTL): The go to local command causes the device currently addressed to listen to return to local panel control (exit the REMOTE state). The device will

return to remote when it is addressed to listen again.

Parallel Poll Configure Command (PPC): The parallel poll configure command causes the addressed listener to be configured according to the parallel poll enable command which will follow.

SECONDARY COMMANDS:

Parallel Poll Enable Command (PPE): The parallel poll enable secondary command configures the devices which have received the PPC command to respond to a parallel poll on a particular GPIB DIO line with a particular level. Some devices may implement a local form of this message (e.g. jumpers).

Parallel Poll Disable Command (PPD): The parallel poll disable command disables the devices which have received the PPC command from responding to the parallel poll.

DATA MODE (ATN false) In this mode device dependent data (e.g. programming data, measurement data, or status data) is sent from the active talker to the active listeners on the interface.

b. END OR IDENTIFY (EOI)

This signal has two uses as its name implies. A talker may assert EOI simultaneously with the last byte of data to indicate end of data. The controller may assert EOI along with ATN to indicate a Parallel Poll. Although many devices do not use Parallel Poll, all devices should use EOI to end transfers (many currently available ones do not).

c. SERVICE REQUEST (SRQ)

This line is like an interrupt: it may be asserted by any device to request the Controller to take some action. The Controller must determine which device is asserting SRQ by conducting a Serial Poll at its earliest convenience. The device deasserts SRQ when polled.

d. INTERFACE CLEAR (IFC)

This signal is asserted only by the System Controller in order to initialize all device interfaces to a known state. After deasserting IFC, the System Controller is the active controller of the system.

e. REMOTE ENABLE (REN)

This signal is asserted only by the System Controller. Its assertion does not place devices into Remote Control mode;REN only enables a device to go remote when addressed to listen. When in Remote, a device should ignore its front panel controls.

3. DATA BYTE TRANSFER CONTROL BUS

3 lines used to coordinate the transfer of data over the data bus from a source (an addressed talker or a controller) to an acceptor (an addressed listener or all devices receiving interface commands) to ensures data transfer integrity. This technique has the following characteristics:

.... Data transfer is asynchronous and the transfer

rate automatically adjusts to the speed of the sender and receiver(s) and runs at the rate of the slowest addressed device.

.... More than one device can accept data at the same time.

.... Every byte transferred undergoes the handshake (except for parallel poll response).

..., When universal commands are sent over the bus, the slowest device on the bus will determine the transfer rate during the transfer of that command.

.... The actual transfer rate of the data is also affected by the time it takes the instrument to take the reading and the time necessary for the controller to input the information.

GPIB signal lines use a low-true logic convention to implement the wired or convention of the NRFD and NDAC lines, provide active true-state assertion, and reduce susceptibility in the true state.

a. NOT READY FOR DATA (NRFD)

This handshake line is asserted by a listener to indicate it is not yet ready for the next data or control byte. Note that the Controller will not see NRFD deasserted (i.e.,ready for data) until all devices have deasserted NRFD.

b. NOT DATA ACCEPTED (NDAC)

This handshake line is asserted by a Listener to indicate

it has not yet accepted the data or control byte on the DIO lines. Note that the Controller will not see NDAC deasserted (i.e., data accepted) until all devices have deasserted NDAC.

c. DATA VALID (DAV)

This handshake line is asserted by the Talker to indicate that a data or control byte has been placed on the DIO lines and has had the minimum specified settling time.

The handshake timing sequence is illustrated in Fig.2.3

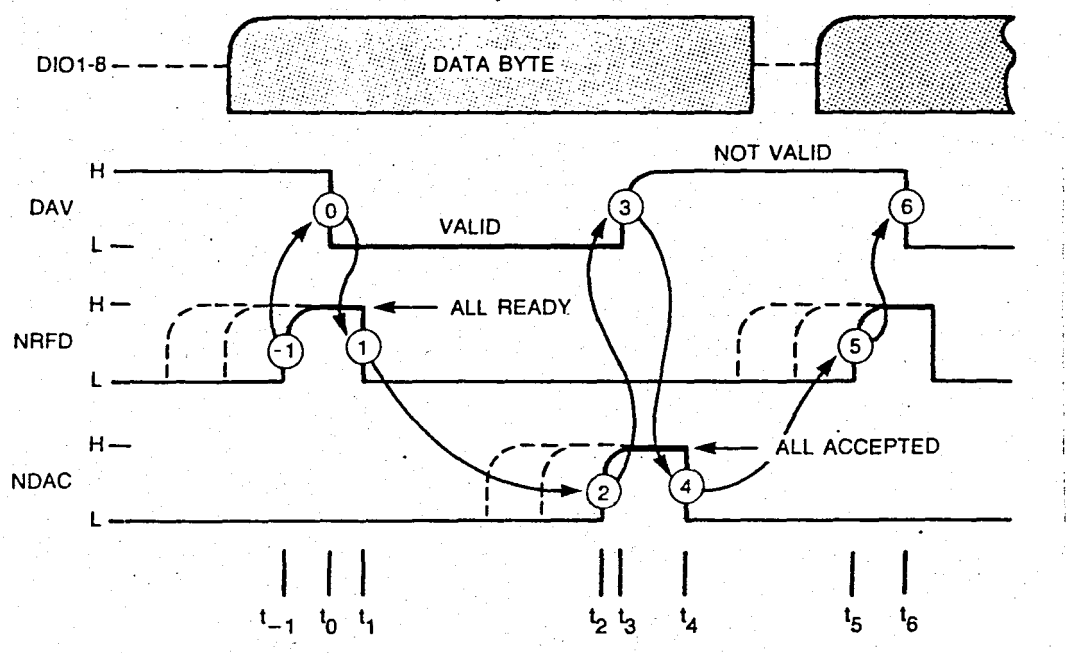


Fig.2.3 Data Byte Transfer

Preliminary: Sourced checks for listeners and places data byte on data lines.

t_{-1} : All acceptors become ready for byte. NRFD goes high with slowest one

t_0 : Source validates data (DAV low)

t_1 : First acceptor sets NRFD low to indicate it is no longer ready for a new byte.

t_2 : NDAC goes high with slowest acceptor to indicate all have accepted the data.

t_3 : DAV goes high to indicate this data byte is no longer valid.

t_4 : First acceptor sets NDAC low in preparation for next cycle.

t_5 : Back to t again.

The handshake sequence is depicted in flowchart form in Fig.2.4

C. GPIB INTERFACE FUNCTIONS

There are ten interface functions specified by the IEEE 488 standard. Not all devices will have all functions and some may only have partial subsets.

1. SOURCE HANDSHAKE (SH):

This function provides a device with the ability to properly transfer data from a Talker to one or more Listeners using the three handshake lines.

2. ACCEPTOR HANDSHAKE (AH):

This function provides a device with the ability to properly receive data from the Talker using the three handshake

lines. The AH function may also delay the beginning (NRFD) or end (NDAC) of any transfer.

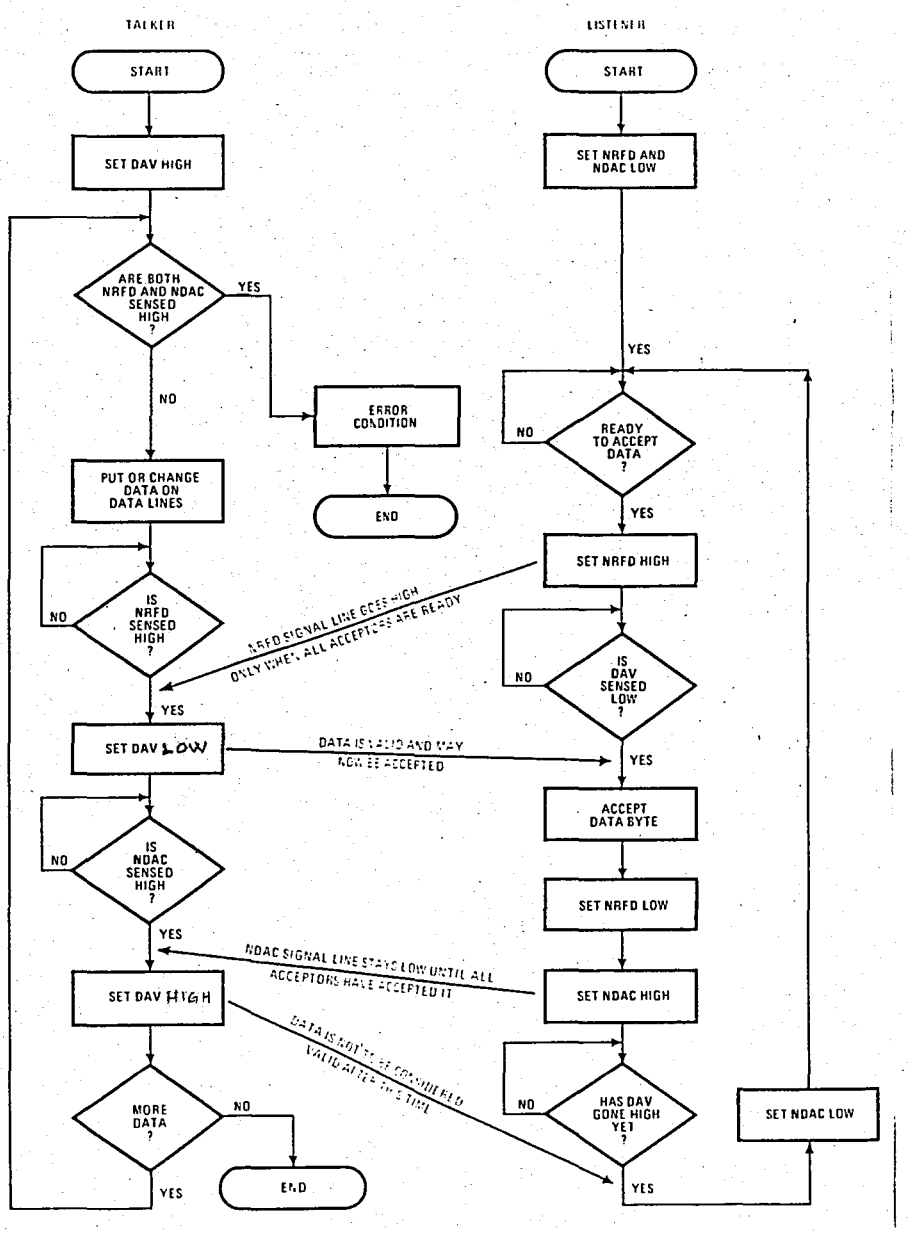


Fig.2.4 Flow chart of data transfer

3. TALKER (T):

This function allows a device send status and data bytes when addressed to talk. An address consists of one (Primary) or two (Primary and Secondary) bytes. The latter is called an Extended Talker.

4. LISTENER (L):

This function allows a device to receive data when addressed to listen. There can be Extended Listeners (anologous to Extended Talkers above).

5. SERVICE REQUEST (SR):

This function allows a device to request service (interrupt) the Controller. The SRQ line may be asserted asynchronously.

6. REMOTE LOCAL (RL):

This function allows a device to be operated in two modes: Remote via the GPIB or Local via the manual front panel controls.

7. PARALLEL POLL (PP):

This function allows a device to present one bit status to the Controller-in-charge. The device need not be addressed to talk and no handshake is required.

8. DEVICE CLEAR (DC):

This function allows a device to be cleared (initialized) by the Controller. Note that there is a difference between DC (Device Clear) and the IFC (Interface Clear) line.

9. DEVICE TRIGGER (DT):

This function allows a device to have its basic operation started either individually or as part of a group. This capability is often used to synchronize several instruments.

10. CONTROLLER (C):

This function allows a device to send addresses, as well as universal and addressed commands to other devices. There may be more than one controller on a system, but only one may be the controller-in-charge at any time.

At power-on time the controller that is handwired to be the System Controller becomes the active controller-in-charge. The System Controller has several unique capabilities including the ability to send Interface Clear (IFC clears all device interfaces and returns control to to the System Controller) and to send Remote Enable(REN allows devices to respond to bus data once they are addressed to listen). The System Controller may optionally Pass Control to another controller, if the system software has the capability to do so.

D. GPIB ELECTRICAL ASPECTS

General: The relation between logic and voltage levels is:

Logic Level	Voltage Level
0 (False)	>= +2.0V (High)
1 (True)	=< +0.8V (Low)

Driver Types:

Open Collector Only	Open Collector or ~Tristate
SRQ, NRFD, NDAC	ATN, IFC, REN, EOI, DAV
DIO1-8 (Parallel Poll Devices)	DIO1-8 (non-Parallel Poll Devices)

~Tristate is useful to reach data rates above 250k bytes/s. Tristate is disabled during parallel poll.

Driver Specifications:

The specifications for drivers shall be as follows:

Low state: Output voltage (three-state or open collector drivers) < +0.5V at +48mA sink current
The driver have to be capable of sinking 48mA continuosly.

High state: Output voltage (three-state) >= +2.4V at -5.2mA
Output voltage (open collector) (see DC load requirements)

Receiver Specifications:

The allowed specification for receivers with nominal noise immunity have to be as follows:

Low state: Input voltage =< +0.8V

High state: Input voltage >= +2.0V

The preferred specification for receivers: To provide added noise immunity, the use of Schmitt-type receiver circuits (or equivalent) for all signal lines is recommended. The specifications for these receivers have to be as follows:

Hysteresis: Vpos - Vneg >= +0.4V

Low state : Negative going threshold voltage Vneg >= +0.8V

High state: Positive going threshold voltage Vpos =< +2.0V

Device DC and small signal AC Load Requirements:

These requirements are summarized and clarified with a typical circuit design:

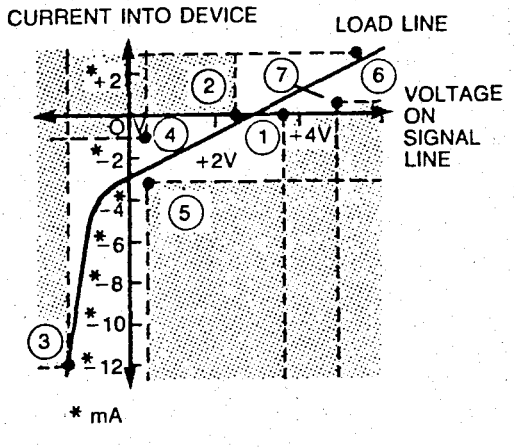


Fig.2.5 I-V Characteristics

Resistive Loading

(1) if I =< 0mA, V shall be < 3.7V

(2) if I >= 0mA, V shall be > 2.5V

(3) if I >= -12.0mA, V shall be > -1.5V

(only if receiver exists)

(4) if V =< 0.4V, I shall be < -1.3mA

(5) if V >= 0.4V, I shall be > -3.2mA

(6) if V =< 5.5V, I shall be < 2.5mA

(7) if V >= 5.0V, I shall be > 0.7mA

or the small-signal Z shall be >= 2k at 1MHz

Capacitive Loading

Cint.=< 100pF at <2.0V

Typical Design:

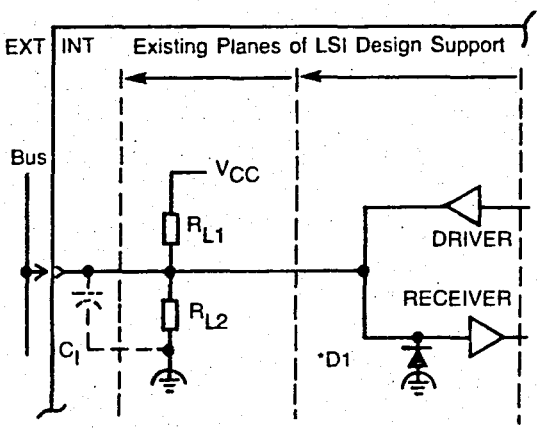


Fig.2.6 Typical design configuration

Vcc : +5V +5%

R_{L1} : 3.1 k +5% (to Vcc)

R_{L2} : 6.2 k +5% (to ground)

Driver : Output leakage current (open collector driver)

+0.25mA max at V =5.25V

Output leakage current (three-state driver)

+40uA max at V =+2.4V

Receiver: Input current

-1.6mA max at V =+0.4V

Input leakage current

+40uA max at V =+2.4V

+1.0mA max at V =+5.25V

$C_1 = C_{cabling} + C_{component} =< 150pF$ at 1kHz

E. GPIB MECHANICAL ASPECTS

The connector, mounting and cabling specifications of the interface define a flexible cabling system for interconnecting IEEE 488 devices. Devices can be interconected in STAR, LINEAR, or combinational arrangements. An overall cabling restriction of 20m total or 2m per device.

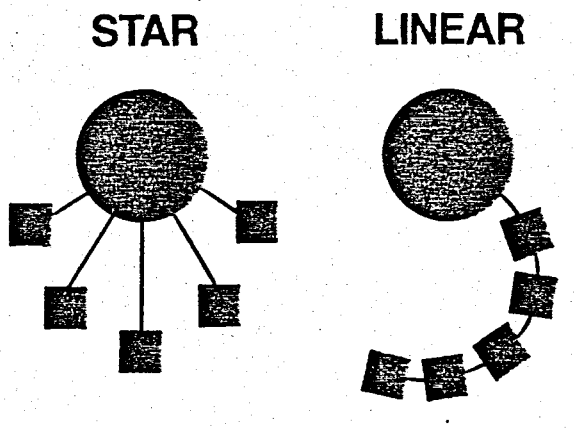


Fig.2.7 Cabling arrangements

Connector: The IEEE 488 connector is a 24-pin ribbon type connector.

Connector Electrical Specification:

Voltage rating	: 200V DC
Current rating	: 5A per contact
Contact resistance	: < 10m
Contact Material	: gold over copper
Insulation resistance	: >10G

Connector Mechanical Specification:

Number of contacts	: 24
Contact surfaces	: self-wiping
Shell shape	: trapezoidal
Shell material	: corrosion resistant material
Endurance	: >= 1000 insertion

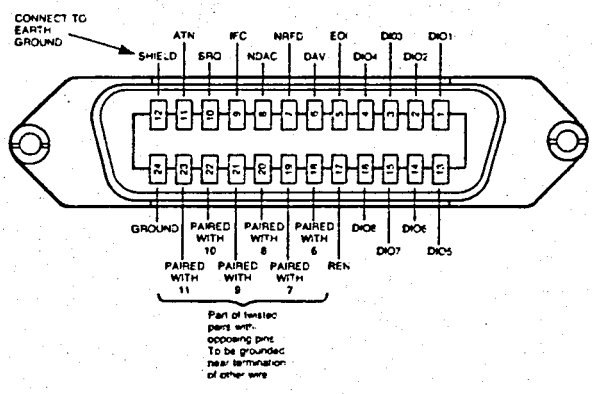


Fig.2.8 IEEE Connector

Connector Pin Description:

Contact	Signal line
1	DIO1
2	DIO2
3	DIO3
4	DIO4
5	EOI
6	DAV
7	NRFD
8	NDAC
9	IFC
10	SRQ
11	ATN
12	SHIELD
13	DIO5
14	DIO6
15	DIO7
16	DIO8
17	REN
18	GND (paired with 6)
19	GND (paired with 7)
20	GND (paired with 8)
21	GND (paired with 9)
22	GND (paired with 10)
23	GND (paired with 11)
24	GND

Note that the grounds of REN and EOI are same and it is pin 24.

III. MESSAGE STRUCTURE IN THE GPIB

A. INTRODUCTION

The message formats which are used in the GPIB are recommended by the IEEE 728 Standard. The objectives of this recommended format are:

1. To promote compatibility among different manufacturers' products

2. To enable the interconnection of instrumentation and releated devices with both limited and extensive capability extensive capability to generate, process, and interpret a variety of different message types

3. To define codes and formats that will minimize the generation of application software and system configuration costs

4. To define a limited family of preferred message codes and formats in a relatively device-independent manner

5. To permit direct communication among instrument system devices without extraordinary translation and conversion special codes and formats

B. SYSTEM CONSIDERATION

The purpose of an interface system is to carry device-dependent messages between devices. According to interface functions, to and from device functions as shown in Fig 3.1 Device functions may differ from one another in several respects, and thus affect the nature of device-dependent

messages that may be interchanged.

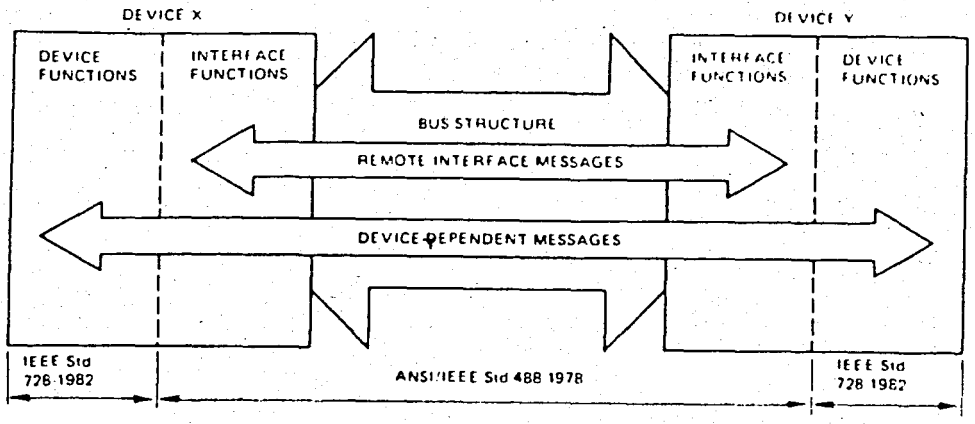


Fig.3.1 Functions and Messages of the Interface System

C. MESSAGE CONCEPTS

1. MESSAGE TYPES AND DEFINITIONS

Devices typically send and receive several different types of device-dependent messages. This section defines a set of message types considered to be useful for a variety of instrumentation system needs. It also defines the terms used to structure the description of these messages.

Measurement Message: A measurement message is a sequence of one or more measurement results seperated by message unit separetors.

A measurement result is the output of an instrument's measurement process. Instrumentation devices typically output measurement information (for example, frequency, voltage, current) as a result of performing a measurement. The general

format of measurement results may contain both an information and a numeric value. A measurement result consists of a sequence of data bytes (DABs) that represent an optional header field and one mandatory data field.

Program Message: A program message is a sequence of one or more program instructions and optional message unit separators.

A program instruction is used to setup and execute an instrument's measurement or stimulus function(s). Instrumentation devices typically receive program data (for example, measurement range, output mode) in preparation for performing a measurement function. A program instruction consists of a sequence DABs that represents at least a mandatory header field and one or more optional data fields.

Status Byte Message: A status byte message carries information about the internal conditions(s) of an instrument's device functions. Devices send a status byte (STB message) in response to a serial poll to indicate one or more device function conditions (for example, measurement complete, out of calibration, current limited). A status byte message is sent with a single status byte.

Display Message: A display message contains information displayed for operator convenience. Display data may contain a series of accumulated measurement results or a series of program instructions used to setup a device. In either case, human interpretation is important.

Message Element: Message elements are the basic building blocks of the syntactic constructs that define program and measurement messages. Major message elements consist of three types of header, four data types, and three types of separators.

2. MESSAGE UNIT ELEMENTS

a. HEADER FIELD

A header field may be used to describe the units (type, quantity) and quality of the data present in the data field it precedes, or both. A header field may also be used to select a specific function.

In all cases, the initial character in an HR field is limited to an alpha character to facilitate parsing (partitioning) of messages by the receiving device. Three header fields available as shown in Fig.3.2 are HR1 an alpha header, HR2 a formatted header, HR3 a character header.

Alpha Header (HR1) : A sequence of one or more alpha characters constructed according to Fig.3.3 Alpha characters may be of either case, upper case preferred.

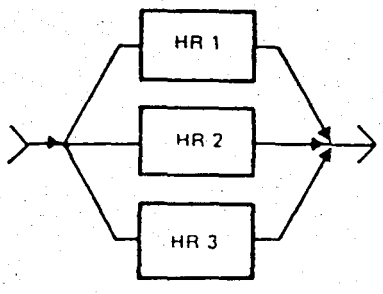


Fig.3.2 HR choices

Fig.3.3 HR1 Syntax Diagram

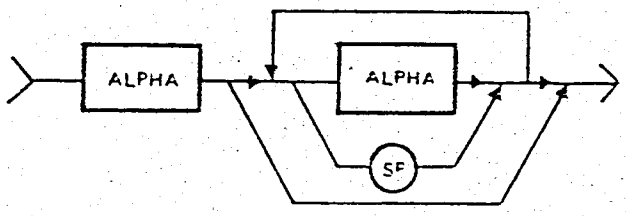


Fig.3.4 HR2 Syntax Diagram

Formatted Header (HR2) : A sequence of one or more alpha characters plus the possibility of embedded or trailing spaces constructed according to Fig.3.4 In some messages it is helpful to maintain the header at a fixed length for a given product employing a variety of type and quality indications. The use of embedded or trailing spaces is therefore useful. Alpha characters are:

A B C D E F G H I J K L M N O P Q R S T U V Y Z

a b c d e f g h i j k l m n o p q r s t u v y z

Character Header (HR3) : A sequence of one alpha character plus the possibility of additional characters following an initial alpha constructed according to Fig.3.5 In the additional character set these printable characters are allowed:

Special characters:

! $ % & ' ( ) * + - / : < = > ? @ [\ ] ^ _ { } ~ ¦

Alpha characters: as shown in HR2

Digits: 0 1 2 3 4 5 6 7 8 9

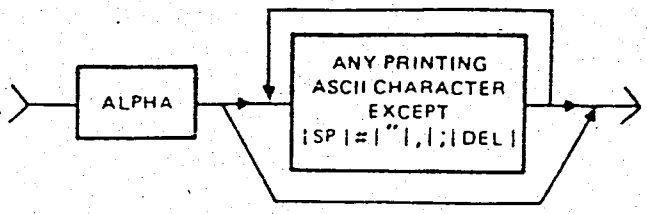


Fig.3.5 HR3 Syntax Diagram

b. DATA FIELDS

The central information contained within the body of each message unit may be represented by using one of four possible data types: numeric, string, block or character.

Numeric Data Type: The decimal positional representation of numeric values, commonly called numeric representation (NR), may be implemented in any of three forms as shown as in Fig.3.6

Numeric digits= 0 1 2 3 4 5 6 7 8 9

Special symbolic representation= E SP(space) + - .

Within an NR field, the most significant digit is sent first.

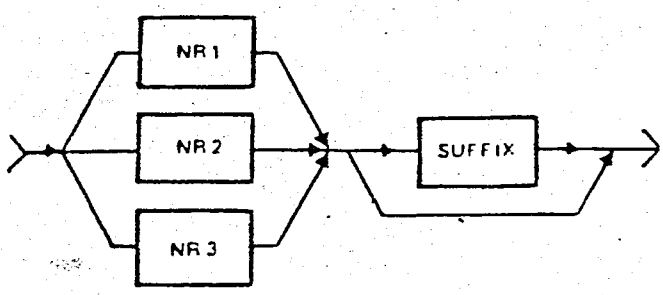


Fig.3.6 Numeric Representation Syntax Diagram

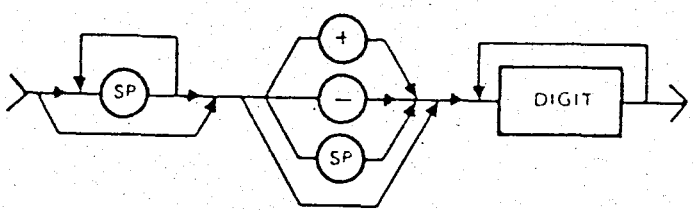


Fig.3.7 NR1 Syntax Diagram

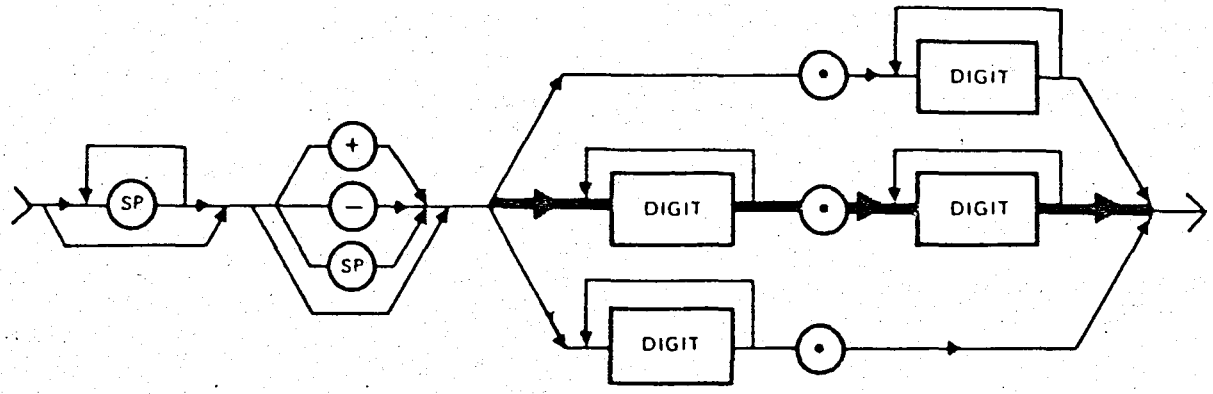


Fig.3.8 NR2 Syntax Diagram

Numeric Representation Set 1 (NR1): NR1 consists of a set of implicit point representation of numeric values, that is, a radix point is implicitly considered to be placed (fixed but not transmitted) at the end of string of digits. Both the signed and unsigned representation may contain leading spaces. The syntax for NR1 is shown in Fig.3.7

Numeric Representation Set 2 (NR2): NR2 consists of a set of explicit point representation of numeric values with the radix point indicated by a decimal point (.) . For clarity the radix point should be preceded by at least one digit, possibly (0).The syntax for NR2 is shown in Fig.3.8

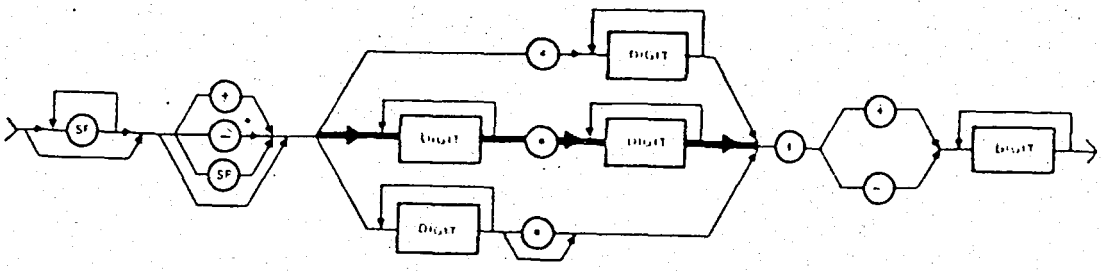


Fig.3.9 NR3 Syntax Diagram

Numeric Representation Set (NR3): NR3 consists of a set of scaled representation with either implicit or explicit radix point together with an exponent notation as shown in Fig.3.9

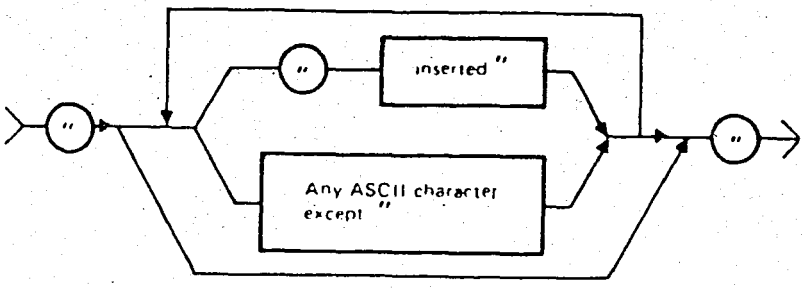


Fig.3.10 String Syntax Diagram

String Data Field:

The string data field allows any character in the ASCII 7-bit code (including non-printable characters) to be carried as a message. This data field is particularly useful where text is to be displayed (for example, on a printer or CRT type device). Using the string data type permits the use of format effectors such as <CR> <LF> <SP> to correctly format text. The string

data syntax is given in Fig.3.10

Block Data Field:

The block data field allows any 8-bit data type (including extended ASCII codes) to be carried as a message. It is particularly useful for sending large quantities of data. Since the focus of the block data field is to provide for long data streams, a means to increase data transfer reliability has been included via error detection. A block data field is initiated by a unique code the number <#> sign. The <#> code is not allowed in any other device-dependent message. The syntax for the block data field is given in Fig.3.11

Block Preamble:

The preamble consists of two bytes the first of which is the <#> byte. A second byte designates which fourteen possible data types are to be represented within the block by the data types.

Block Length Bytes:

The length bytes contain information about the number of bytes in the block. The number of length bytes are 2.

Block Check Bytes:

When a check byte(s) is used the check is performed on only the data bytes (DABs). If a checksum error detection means is used then the check byte shall contain the modulo 256 checksum. If a CRC 16 Forward errror detection means is used, then the two byte count sent with the data bytes contains the remainder from division of the data byte stream by the CRC 16

Forward polynomial (that is $x^{16} + x^{15} + x^{2} + 1$). The receiving device remainder is then zero for an error-free message.

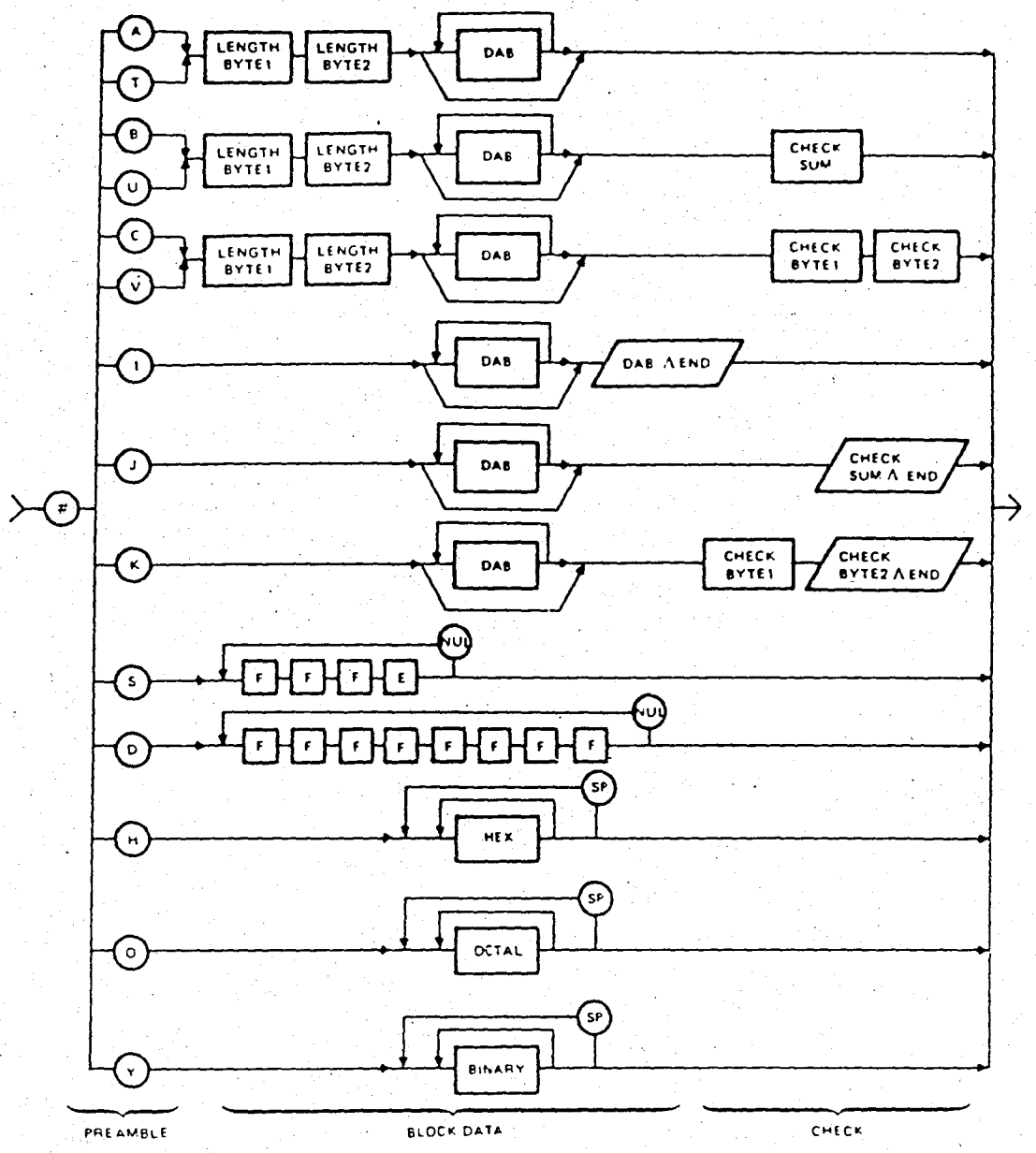


Fig.3.11 Block Data Field Syntax Diagram

Character Data Field:

The character data field is used where words and text more clearly describe the nature of a program instruction than does a numeric data field. Character data fields always begin with an alpha character. The use of alpha characters only is preferred. Digits and other special purpose characters should be used only where human readibility and interpretation of the header is thereby enhanced. The character data syntax is shown in Fig.3.12

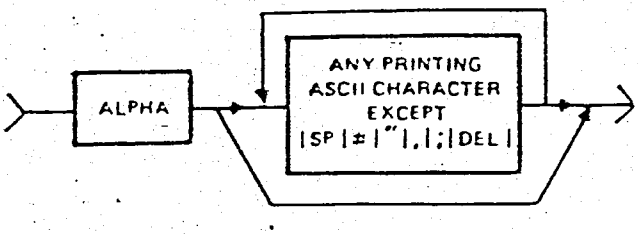


Fig.3.12 Character Syntax Diagram

c. MESSAGE SEPARATORS

A means to distinquish between one message unit and the next message unit or between information fields within message units is useful and sometimes essential for unambiguous message processing. Such means are useful for conveying related sets of data that occur in pairs (for example , amplitude and phase) or other data set multiples (for example , time , channel , frequency) or long continuos message streams. The separators listed in descending order of the hierarchy are SR3, SR2, SR1 A very general structure for Separator usage is shown in Fig.3.13

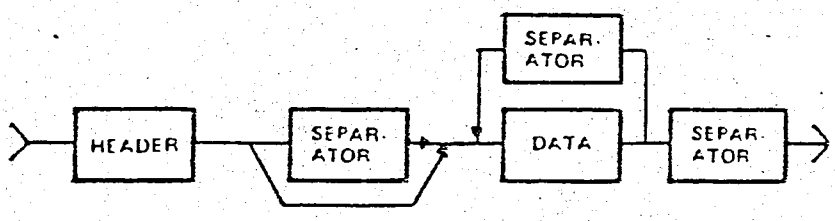


Fig.3.13 Typical Separator Use

Separator Level 1 (SR1): An SR1 separator iis typically used to identify the end of the lowest level of message element or data field(s). Two separators exist at this level: the comma <,> and semicolon <;>. The lowest order separator of these two is the <,> and is for most applications, the preferred separator. This based on the fact that the comma <,> is of lower than the <;> in the written word. The syntax for SR1 is given in Fig.3.14

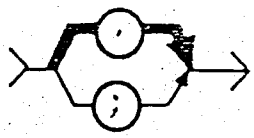


Fig.3.14 SR1 Syntax Diagram

Separator Level 2 (SR2): An SR2 separator is typically used to separate a sequence of message units at a distinctly higher level than that of the SR1 separators. There is only one level of at the SR2 level although there are several implementation choices. The syntax of SR2 is given in Fig.3.15

Separator Level 3 (SR3): The SR3 separator is the

highest order separator. An SR3 separator is usually used when one or a series of measurement or program messages has been completeted. As the highest order separator, the SR3 or END message has special significance. A talker, having sent END, shall not output further DABs automatically. The talker must then receive a device-dependent message or a specific interface (for example GET-Group Execute Trigger-) prior to resuming output. The syntax of SR3 is given in Fig.3.16

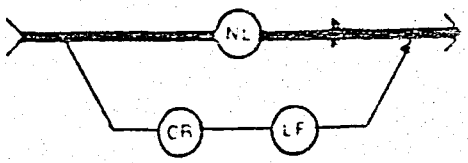


Fig.3.15 SR2 Syntax Diagram

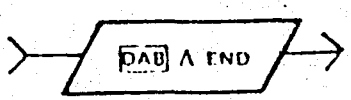


Fig.3.16 SR3 Syntax Diagram

3. MEASUREMENT MESSAGES

Instrumentation devices perform many measurement functions. Typically tasks measuring frequency or voltage, digitizing a waveform, and performing network analysis result in the generation of measurement results. The specific content of the message is device-dependent, however the organization or general format of these application dependent messages can be structured in a way comman to many devices. The purpose of this

section is to define the format of measurement messages.

Measurement Message Formats:

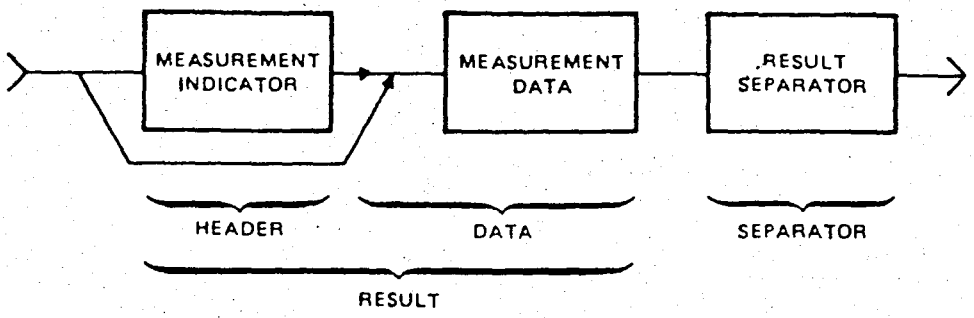


Fig.3.17 General Form of Measurement Message

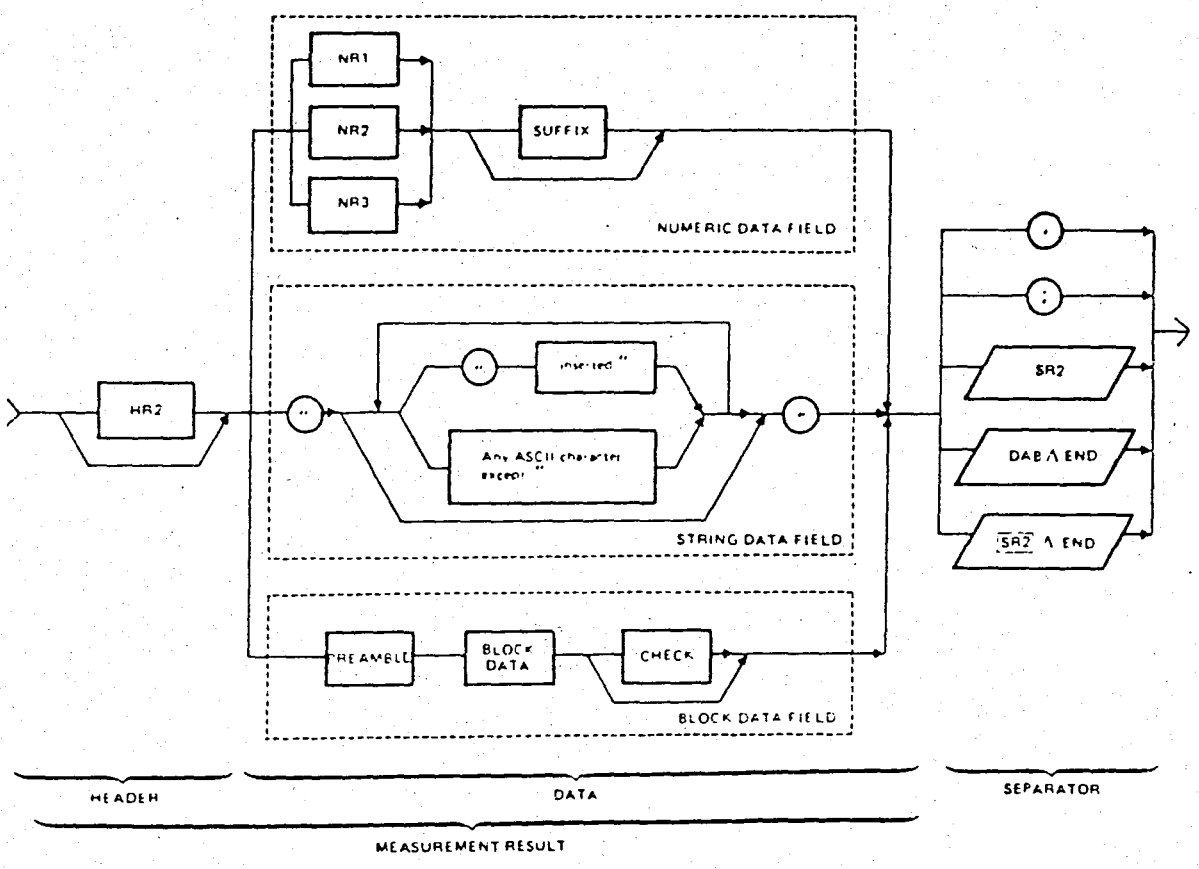


Fig.3.18 Measurement Message Syntax Diagram

A generalized form of typical measurement messages is given in Fig.3.17 to introduce the normal flow of data fields, the format of the message. There are three different types of measurement messages. The Fig.3.18 shows the syntactic structure of the three different types of measurement results identified as numeric data, string data, and block data.

4. PROGRAM MESSAGES

Instrumentation devices are called upon to perform many different tasks. This process of performing different tasks requires changes in the basic measurement or stimulus task. A device might be required to changed to change operating mode, measurement range, or output mode, for example. Receipt of program instructions effect the necessary changes. The specific content of this data is device-dependent, however the organization or general format of these application dependent messages can be structured in a way common to many devices. The purpose of this section is to define the format of program messages.

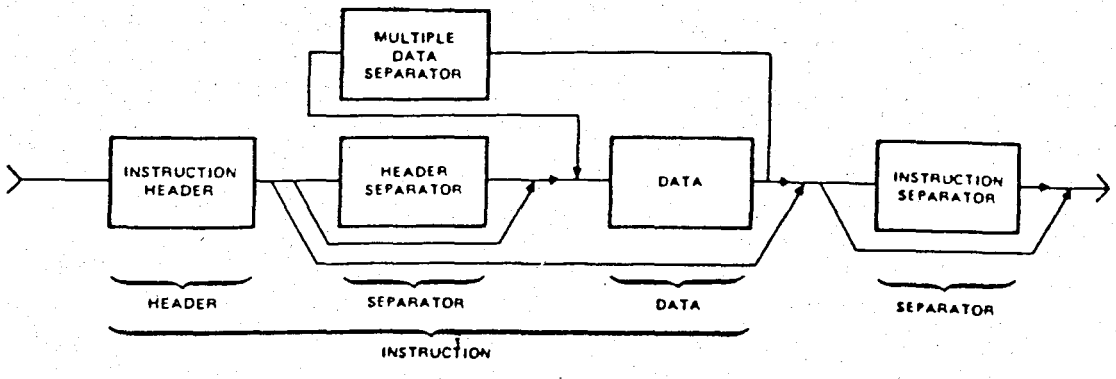


Fig.3.19 General Form of Program Message

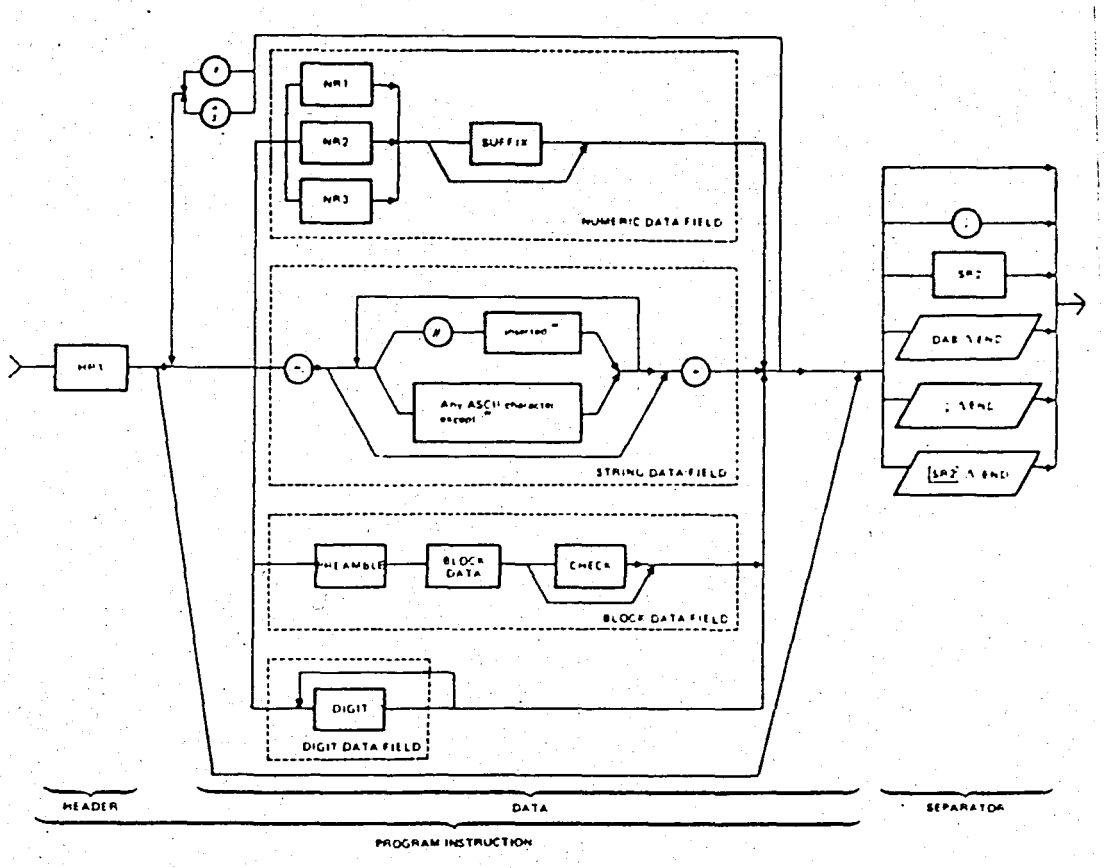


Fig.3.20 Program Message Syntax Format

Program Message Formats: A generalized form of typical program messages is given in Fig.3.19 to introduce the normal flow of data fields, the format of the message. Fig.3.20 illustrates the syntactic construction of four different types of program insructions identified as digit data, numeric data, string data, and block data.

5. STATUS MESSAGES

A status message may be sent from a device with an STB message in response to a serial poll sequence when the device is

in the serial poll active state (SPAS). The principle purpose of the STB message is to present critical summary and detailed status to the controller-in-charge. Summary status means the logical OR or detailed status within the same category, for example, if more then one abnormal condition exists. A wide variety of device-dependent internal states and conditions may be carried by STB messages. Therefore a complete data structure code assignment is not feasible.

6. DISPLAY MESSAGES

There is no specific format for the syntax of data uniquely utilized for display purposes. Both the program and measurement messages, however can be used for display purposes.The character and string data types are particularly useful for this purpose. Display messages are intended for device input or output where human interpretation is important.

7. ERROR DETECTION AND CORRECTION

The need for error detection capability within instrumentation systems varies significantly in relation to the nature of the noise environment, nature and importance of data carried on the interface active at both data source and acceptor, and overall system application. There are several common techniques for error detection and their applications are device-dependent. Some of them are as listed below along with comments about their applications

The error detection and correction are not the main subject of this work, therefore these methods are explained as an information.

Error Detection "at the Bit Level":

A simple lateral parity bit on DIO8 to detect errors contained in DIO1-7 for the ASCII 7-bit code provides minimal means for error detection and requires minimal hardware. Parity check permits detection of a single error within the bit grouping of a byte. Multiple-bit errors within a byte may not be detected.

A longitudinal parity check bit on any given DIO line at the end of data record may be used in the same way as the lateral parity check bit for the same purpose on results.

A cyclic redundancy check (CRC) is much more comprehensive and assures a higher degree of error detection capability.

Error Detection "at the Message Level":

An instrument receiving a programming command message can check the message for proper syntax so that all the data that is expected has the proper format. Commands not having the proper syntax should not be executed. Instead, a service request (SRQ) message should be generated and the status byte (STB) on the subsequent serial poll should indicate a syntax error.

An instrument receiving a programming command message can check the semantics of the message to see if they are meaningful and executable. For example, a power supply with a maximum

output of 50V should not attempt to execute a command that tells it to output 500V. Instead, a service request should be generated and the status byte on the subsequent serial poll should indicate an execution error.

The types of errors listed in the above paragraphs are generally human errors in programming instruments or controller generated commands which produce out of bounds conditions. The recovery from these errors normally involves human intervention and is beyond the scope of these recommended practices.

Error Correction:

Error recovery is often a required capability in systems having error detection. Specific error recovery techniques are beyond of the scope of this thessis work. Two common approaches are retransmission of data received in error and use of forward error correction (for example, Hamming Codes).

IV. HARDWARE DESCRIPTION

A. GENERAL VIEW TO CONTROLLER

The controller is made by using an INTEL 8085 microprocessor at working 6.144MHz clock rate. The controller is connected to a intelligent monitor via an RS 232C interface circuit. There is also another RS 232C interface circuit for printer. The later one can be used for communication with another computer by a little changing in software. There is a 24-pin parallel port for parallel communication. GPIB interface is also available.

The memory capacity of the controller is 64k byte. The 32k byte of this area is reserved for code of controller, and 16k byte part (divided into two 8k byte area) is used. The other 32k byte area is divided into two 16k byte areas for RAM and EEPROM, but realized 4k byte of them. Available EEPROM and RAM areas are 2k bytes.

B. MECHANICAL DESCRIPTION

Phsycal dimension: The dimension of the contoller card is 160mm x 320mm

Connectors: The controller card contains 4 connectors. The two of them are standart D-type 25-pin connector for RS 232C interface used for printer and monitor communication. One of the connectors is 50-pin flat cable connector for parallel port connection. The last one is the standard type 24-pin ribbon

connector for GPIB.

Glossary of Designation:

A-Bus... Address Bus (A0..A15)

AD-Bus.. Multiplexed address and data bus (AD0..AD7)

D-Bus... Data bus (D0..D7)

C-Bus... Control Bus

TRAP.... Nonmaskable interrupt of microprocessor

RST 7.5 Restart 7.5 interrupt of microprocessor

RST 6.5 Restart 6.5 interrupt of microprocessor

RST 5.5 Restart 5.5 interrupt of microprocessor

INT..... Interrupt of the microprocessor

CLK..... Clock output of the microprocessor

SCOMBR.. Serial communication baud rate clock

TxD/RxD. Input/Output of the USART

C. ELECTRICAL CIRCUIT DESCRIPTION

1. MICROPROCESSOR AND ITS SUPPORT CIRCUITRY

a. MICROPROCESSOR

The controller has been designed by using an INTEL 8085 microprocessor (U1) to execute all logical functions of the Controller. This 8085 is a 8-bit microprocessor with 64k byte memory, 256 byte I/O addressing capacity. It is operating at 3.072 MHz speed without taken into HOLD state.

b. CRYSTAL

6.144 MHz crystal is used for the microprocessor timing.

c. RESET CIRCUITRY

The reset circuitry consists of a diode (1N4148), a resistor (100k), and a capacitor (1uF). This circuit serves to RESET the microprocessor at power on. RESET IN input of the microprocessor stays at logic high level for a while. The capacitor of the reset circuitry charges to logic high level via the resistor. Therefore RESET OUT pin of the microprocessor stays at logic low level. RESET timing is dependent on the time constant of capacitor and resistor. Charge duration of the capacitor from 0V to 3.6V is approximately 128msec. Discharge duration of this circuitry from 5V to 2V is approximately 100msec.

d. MICROPROCESSOR INTERFACE CIRCUIT

U3 (74 LS 373) octal latch is used for this purpose. It is used to separate multiplexed AD-Bus by using address latch enable (ALE) signal coming from microprocessor to enable of U3. When ALE is high it indicates that the signals (A0..A7) at AD-Bus are lower address signals (A0...A7) and these signals are transmitted to the output of U3. U3 is never tri-stated because its output control pin is grounded. When ALE is low, signal at AD pins of the microprocessor are data signals and taken from the inputs of U3 and used as D0-D7. U2 (74 LS 373) is driver circuit for higher bits of the address bus. U4 (74 LS 245) is bidirectional buffer for data bus.

2. MEMORY AND ITS SUPPORT CIRCUITRY

a. MEMORY DECODER

Memory decoder consists of U5 (74 LS 138) and U6 (74 LS 139). A15,A14,A13 lines of A-Bus are fed as 3 addressing bits of U5, so 64k byte memory are is divided into eight of 8k byte areas. The first two output of U5 is used for selecting of U7 and U10. In order to address the other four memory chips, U6 is used. It contains two 2-to-4 decoder. The fifth output of U5, A11 and A12 bits of A-Bus are used to divide 8k byte memory area into four 2k byte memory areas. So two outputs of the first part of U6 are used to select two EEPROM chips. The second part of U6 is used to select two RAM chips by using eigth output of U5, A11 and A12 buts of A-Bus.

b. MEMORY CHIPS

The memory of controller is consists of 6 memory chips. These are:

2 x 8k byte EPROM U7, U10 (2764)

2 x 2k byte EEPROM U8, U11 (X2816)

2 x 2k byte RAM U9, U12 (6116)

one of the each item is optional. It will be used in future if necessary.

c. MEMORY MAP

Selected IC	Address
U7 EPROM 1 (2764 8k byte)	0000H.....1FFFH
U10 EPROM 2 (opt.) (2764 8k byte)	2000H.....3FFFH
U8 EEPROM 1 (X2816 2k byte)	8000H.....87FFH
U11 EEPROM 2 (opt.) (X2816 2k byte)	8800H.....8FFFH
U9 RAM 1 (6116 2k byte)	F800H.....FFFFH
U12 RAM 2 (opt.) (6116 2k byte)	F000H.....F7FFH

3. I/O PORTS

a. I/O PORT DECODER

U13 (74 LS 138) 3-to-8 decoder is used to select necessary ports. A7, A6, A5 bits of A-Bus are fed as # addressing bits of U13. All I/O ports are selected as I/O mapped I/O. U13 selects seven ports; these are U14 (8253 Timer), U16 (8251 USART), U17 (8251 USART), U18 (8291A GPIB Talker/Listener), U19 (8292 GPIB Controller), U27 (8255 Parallel port), U33 (74 LS 373 8-bit Input port). E1 input of U13 is fed by IO/M output of microprocessor. E2A and E2B inputs

are pull down.

b. TIMER

This unit consists of two chips U14 (8253) and U15 (74 LS 74). U15 is a two D-Flip Flop and used one of them to divide 3.072MHz clock out signal of microprocessor for providing necessary clock signals of 8253's counters. U14 is a programmable interval timer counter. Its function is that of a general purpose multi-timing element. It is treated by the system software as an array of peripheral I/O ports three are counters and fourth is a control register for mode pragraminnig. Basically the select inputs A0 and A1 are connected to A0 and A1 lines of A-Bus respectively. CS is connected to the seventh output of U13. One of the counters is used to generate baud rate clock of USART's (U16, U17 8251). The other one is used for RST 7.5 interrupt of 8085. And the last one is spare.

c. SERIAL INPUT/OUTPUT COMMUNICATION UNIT

This unit consists of two USART chips (8251) U16 and U17. They are operated at asynchronous mode and at x64 mode. Therefore SCOMBR clock which is obtained from U14 is divided by 64 internally. One start, one stop, one parity bit added to the actual 8-bit data by the USART automatically. By programming U14 (8253) the baud rate can be changed, because SCOMBR is getting from U14.

For printer interface TxD, DTR, and RTS outputs of U16

is fed to RS232C transmitter chip U22 (MC 1488), U23 (MC 1489) RS232C receiver chip is fed to RxD, DSR, and CTS input of U16.

For monitor just TxD and RxD pins are used with U24, U25 RS232C interface chips.

d. PARALLEL INPUT/OUTPUT PORT

U27 (8255) is a three input/output 8-bit port. These ports are selected by I/O decoder, A0 and A1 bits of A-Bus.

U33 (74 LS 373) is another input port in order to input of the interrupt output of the 8292' TCI (Task completed Interrupt). The other 7 bits of the port are spare.

e. IEEE 488 BUS INTERFACE UNIT (GPIB)

This unit is consists of four chips U18, U19, U20, and U21.

U19 (8292) is a IEEE 488 Bus (GPIB) controller interface element. It is used with the 8291A GPIB Talker/Listener and two 8293 GPIB Transcievers to form a complete IEEE 488 Bus interface for a microprocessor.

The interconnection of these four chips are made as the recommended form in which the data sheets of these chips, in order to implement full TALKER, LISTENER, CONTROLLER functions

The microprocessor accepts them as I/O ports. 8292 has 6 read register and 5 write register. Each one seems to the microprocessor as distinct ports. Also 8291A has 8 read register and 8 write register as the same form. The selection

of these registers is provided by the address lines. A0 line for 8292 to select command registers and operation registers. A0,A1,A2 lines for 8291A to select one of the 8 different registers

The chip selects of 8292 and 8291A come from the I/O decoder. Read and Write inputs are connected to the same outputs of the microprocessor. The only different is that, the reset input of the 8292 accepts inverted reset signal, so this is provided by using an inverter to reset out of the microprocessor and connected to the 8292's reset input.

The interrupt output of 8291A and OBFI, IBFI, SPI interrupt outputs of 8292 are not used. Just the TCI interrupt is used by connecting to the U33 (74 LS 373). The microprocessor checks this port to learn the status of the controller after or before giving a command to the 8292.

d. I/O MAP

Selected IC	A7	A6	A5	A4	A3	A2	A1	A0
U18 8291 A	0	0	0	0	0	X	X	X
U19 8292	0	0	1	0	0	0	0	X
U27 8255	0	1	0	0	0	0	X	X
U33 74 LS 373	0	1	1	0	0	0	0	0
U16 8251	1	0	0	0	0	0	0	X
U17 8251	1	0	1	0	0	0	0	X
U14 8253	1	1	0	0	0	0	X	X

X means that bits can be take the value of either 0 or 1 according to port or register to be choosen.

V. SYSTEM SOFTWARE STRUCTURE

A. INTRODUCTION

UMSC system is a microprocessor based controller system. The main tasks of the UMSC are execution of user program and control of GPIB Bus properly. In order to realize them, some sub-modules are necessary. Those are;

... An editor to write user program into the nonvolatile memory.

... A list function to display written program on to the CRT monitor.

... A syntax checker to control the user program instructions whether they are written correctly or not. At the result of this check process, The program displays the errors and their locations in the line, if exist.

... An interpreter to execute the correctly written user programs

The mentioned sub-modules are activated by the special characters and each time only one routine can be activated In this way, no confusing occurs because of using the same data structures.

All of these tasks are not time critical, so they are activated on the background cycle.

By the execution of interpreter some measurement results or some test comments or headings might be written on the printer. It is obvious that printers are slower devives, if

the controller waits the printer until the printer finishes desired printout, this is waste of time. In order to solve this problem, the controller uses a printer buffer and formats it with desired output strings or/and measurement result by using ASCII characters, and leaves the buffer to the printer sender routine which activated by foreground scheduler. So the controller does not spend time while the printer is working.

Another background sub-module is Monitor. This is for program development and debugging tasks. There is no relation between the mentioned tasks in above.

All the sub-modules including the Printer Sender routine have state driven structure.

According to the above definitions of the software structure of the UMSC, the system software can be divided into two parts. These parts;

... Background S/W

... Foreground S/W

B. SOFTWARE DESIGN HIGHLIGHTS

.... Programming languages are PLM80 and ASM80

.... S/W design is structured and modular

.... Organized as two lewels, background and foreground

.... Requires only one interrupt input which is used to activate the foreground.

C. ARCHITECTURE

The software structure of controller is shown in Fig.5.1. After a power-on reset or an operational reset the program jumps to initialization routine. At the end of this routine microprocessor interrupt input is enabled. After that process the program passes the control to Master control routine which searches continuously the keyboard input routine. According to the input character, Master control branches to the desired sub-module if the pressed key is a valid key.

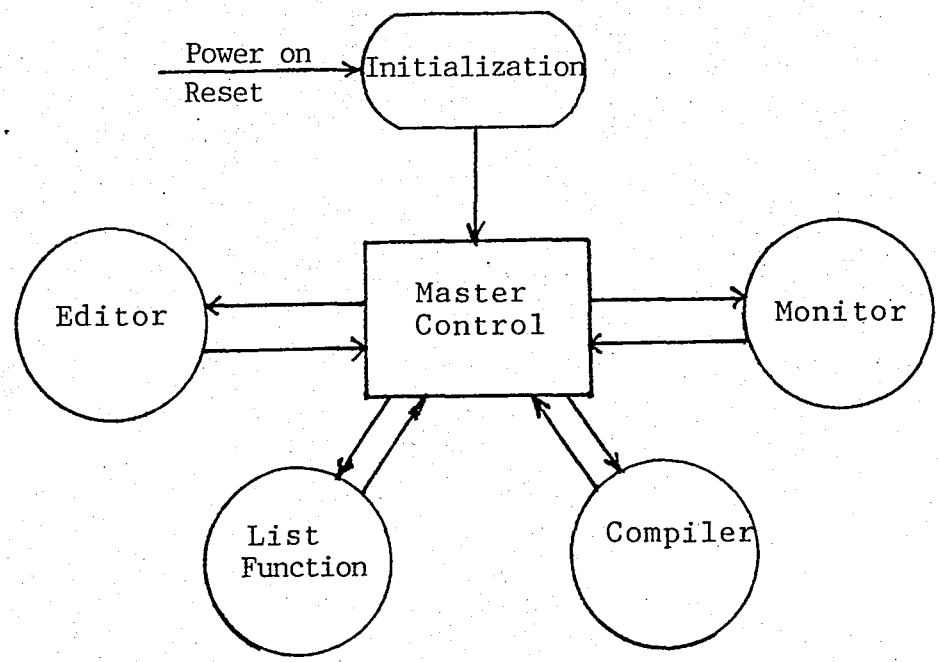


Fig.5.1. General structure of controller

1. DATA STRUCTURES

EDFIL: 16 Bytes

EDSTA : State entry

COLUMN: Column pointer

LINENO: Line pointer

KEY : Pressed key buffer

IDLCTR: Empty line counter

REDSTA: Return state entry

RI : Counter

FIRST : First line number buffer

LAST : Last line number buffer

ENDFLG: End flag for interpreter

JMPFLG: Jump flag for interpreter

EDSPR : 5 Bytes spare

WBUFER: 32 Bytes

Key collectig buffer for editor

CONV: 6 Bytes

Binary to ASCII conversion buffer

DISBUF: 33 Bytes

Display buffer to send characters to the CRT monitor

SYSFIL: 16 Bytes

PRTCTR: Counter for 1.25 msec interrupts

GPBCTR: Not used

SYSCTR: Not used

SYSSPR: 13 Bytes spare

```
        PRTFIL: 16 Bytes
STAFL : Status file of printer output buffer (6 bytes)
LINPTR: Line pointer for printer sender
COLPTR: Column pointer for printer sender
DSRCNT: Printer USART DSR status counter
FAILCT: Printer fail counter
CARFLG: <CR> flag for line
LPCNT : Empty printer buffer counter
PRTSPR: 4 Bytes spare
        OBUFER: 6*80 Bytes
Printer output buffer
        INBUF: 32 Bytes
GPIB input buffer
        REG: 6*32 Bytes
6 software register for GPIB input
        OBUF: 28 Bytes
GPIB output buffer
        OBFCNT: 1 Byte
Counter of elements in the OBUF
        VARA: 2 Bytes
"K" variable
        VARB: 2 Bytes
"L" variable
        LLIST: 2 Bytes
GPIB listener list buffer
```

TLIST: 2 Bytes

GPIB talker list buffer

MONFIL: 48 Bytes

Monitor workspace

2. INITIALIZATION

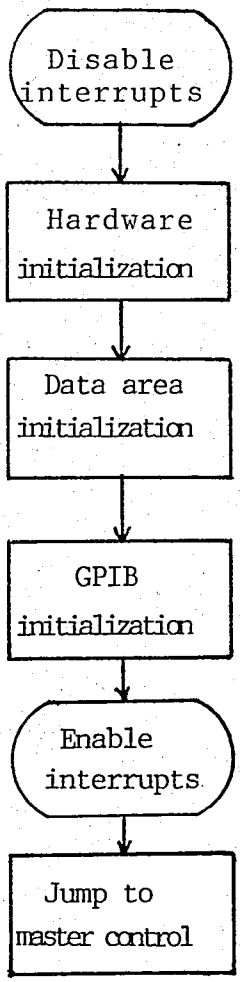


Fig.5.2 Flow chart of Initialization routine

When the system reset by external reset switch or

power-on reset, the program jumps to the initialization routine, and initializes the hardware and data area according to the desired form. Before pass the control to the master control routine, enables the RST7.5 interrupt input, to activate the foreground. The flow chart of this routine is given Fig.5.2.

INIHW initializes the serial ports to the communicate at 1200 baud, and also initializes the timer to obtain necessary clock input for serial ports and foreground activation interrupts to microprocessor at the value of 1.25 msec.

3. MASTER CONTROL

After initialization, the system is ready and waits at standby. It contiunously polls the CRT input routine. If any key is pressed from keyboard, it takes the key and searches whether it is valid or not. According to the result of this search, the program passes the control to chosen module if the key is valid, or displays an error message on the CRT monitor if the key is invalid. The flow diagram of Master control is given in Fig.5.3.

4. EDITOR

This module has a state driven key activated structure. It writes a pressed valid key into the nonvolatile memory which is divided 60 lines each has 32 bytes.

The line number is the first entry of this routine. After taking this entry, program searches the given line and

displays its content on to the CRT monitor.

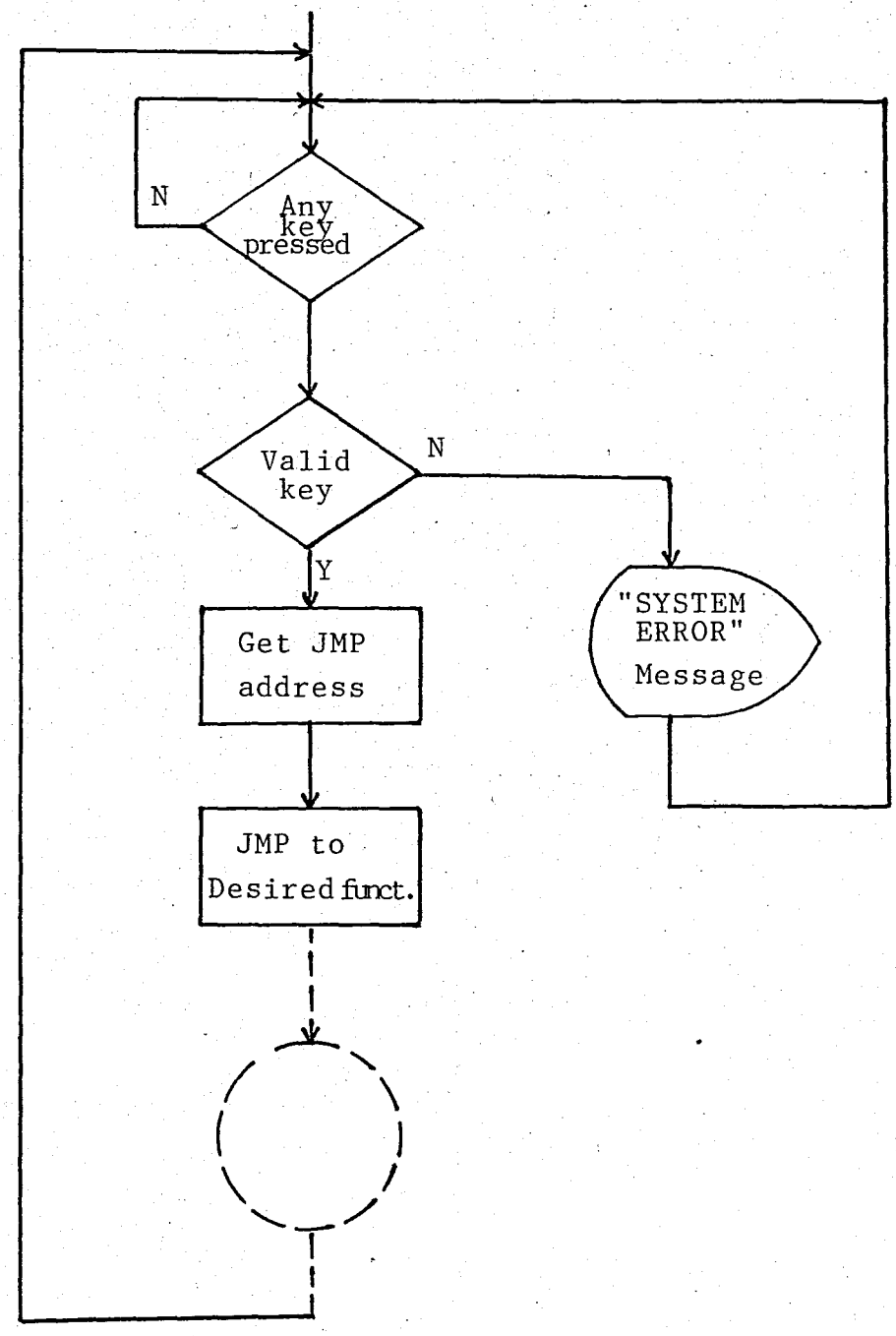


Fig.5.3 Flow chart of Master control routine

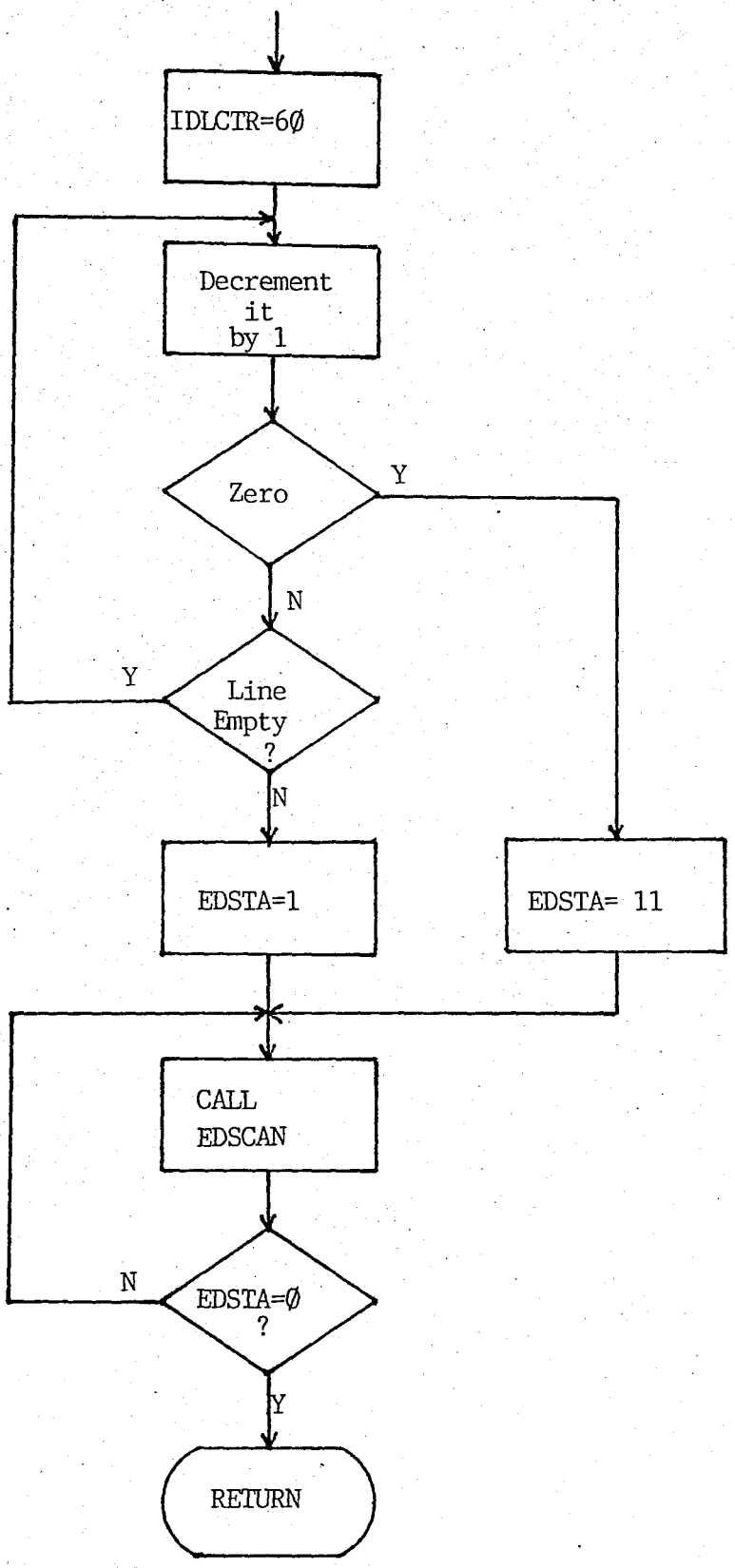


Fig.5.4 Flow diagram of EDCMD routine

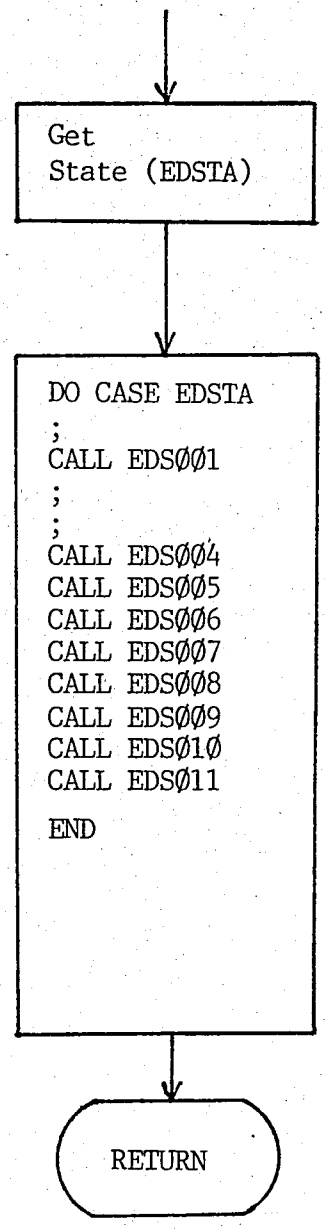


Fig.5.5 Flow diagram of EDSCAN routine

The control characters of this routine are;

<ESC>, <DEL>, <CR>, <BS>, <Right Arrow>

<ESC>.. Edit current line and exit from editor

<DEL>.. Delete current line and continue to next line

<CR>... Finish the current line

<BS>... Go one character back

<Right arrow>... copy one character to buffer from line

While the typing line number, these control characters have similar effect except that any mistake turns the user to the start state.

While edition, the editor picks and puts characters into a RAM buffer (WBUFER). After <CR>, WBUFER is moved to nonvolatile memory byte by byte. The interval of between the transfer of two bytes has to be minimum 10msec. Because the writing a byte into the nonvolatile memory (EEPROM) needs this delay The flow diagrams of editor routine are given in Fig.5.4 and Fig.5.5.

a. DESCRIPTION OF EDITOR STATES

EDS000: Editor exit state

EDS001: Editor enter and preparation of getting line number state.

EDS004: Getting first digit of line number state.

EDS005: Getting second digit of line number state.

EDS006: Preparation state for line edition.

EDS007: Character collection state.

EDS008: Transfer from RAM buffer to EEPROM byte by byte.

EDS009: Messages to user that the no more empty line.

EDS010: Clears all lines if no empty line, if desired.

EDS011: Enter state to editor when all lines are full.

The state diagram of editor is given Fig.5.6.

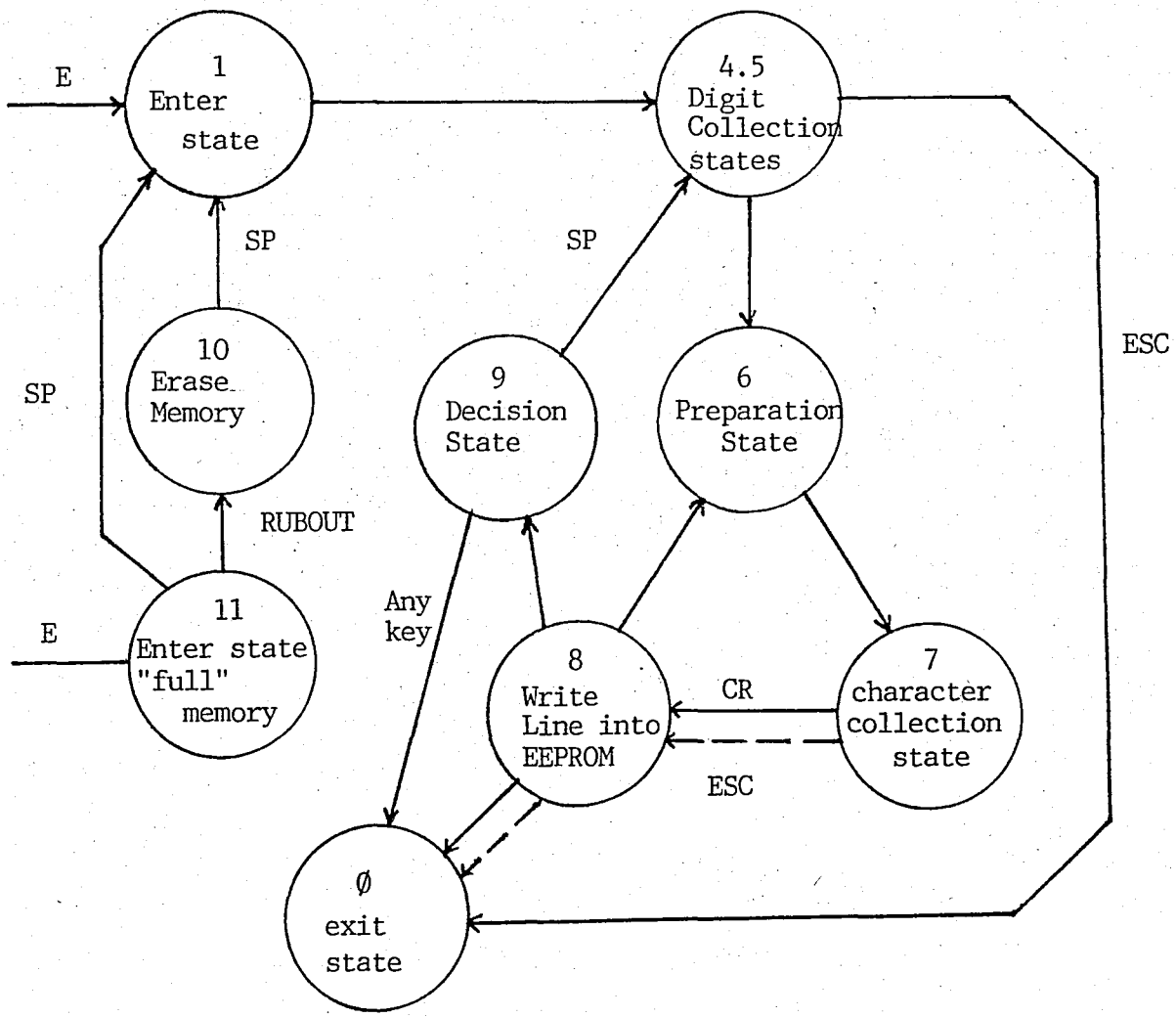


Fig.5.6 State Diagram of EDITOR module

5. LIST FUNCTION

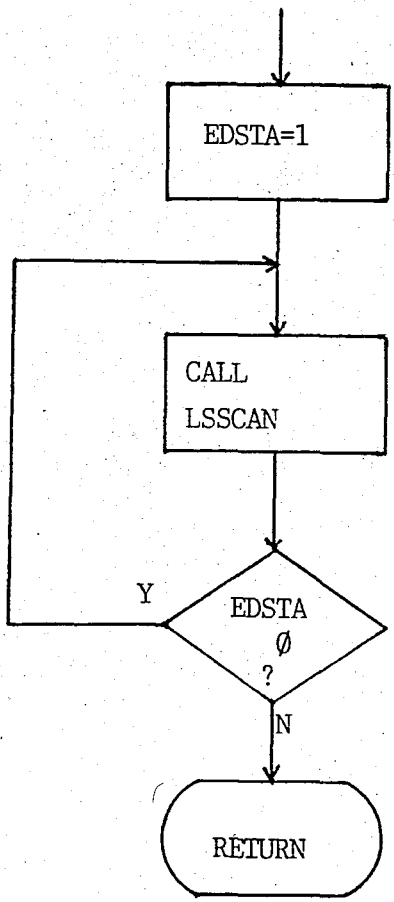


Fig.5.7 Flow diagram of LSTCMD routine

This module is for displaying the lines on to the CRT sequentially from given first line number to last line number.

It is similar to editor, except that, the editor wants one line number, the list routine wants one or two line number. Again it has state driven and key activated structure. It uses the same data structures and temporary data areas.

The list routine is activated from master control, if the user presses the <L> key on standby. The program jumps to the LSTCMD where the state number is given "1" (Enter state number of LIST routine). After that, LSTCMD makes a loop while the state is different from "0". If it is so, program returns to master control. The flow diagrams of list routine are given in Fig.5.7 and Fig.5.8.

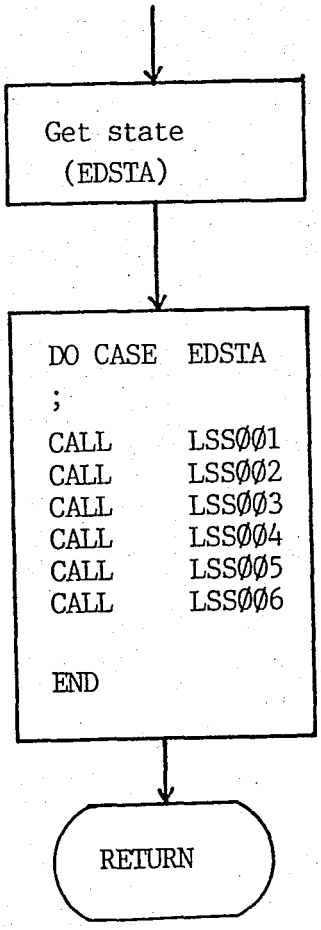


Fig.5.8 Flow diagram of LSSCAN routine

a. STATE DESCRIPTION OF LIST FUNCTION

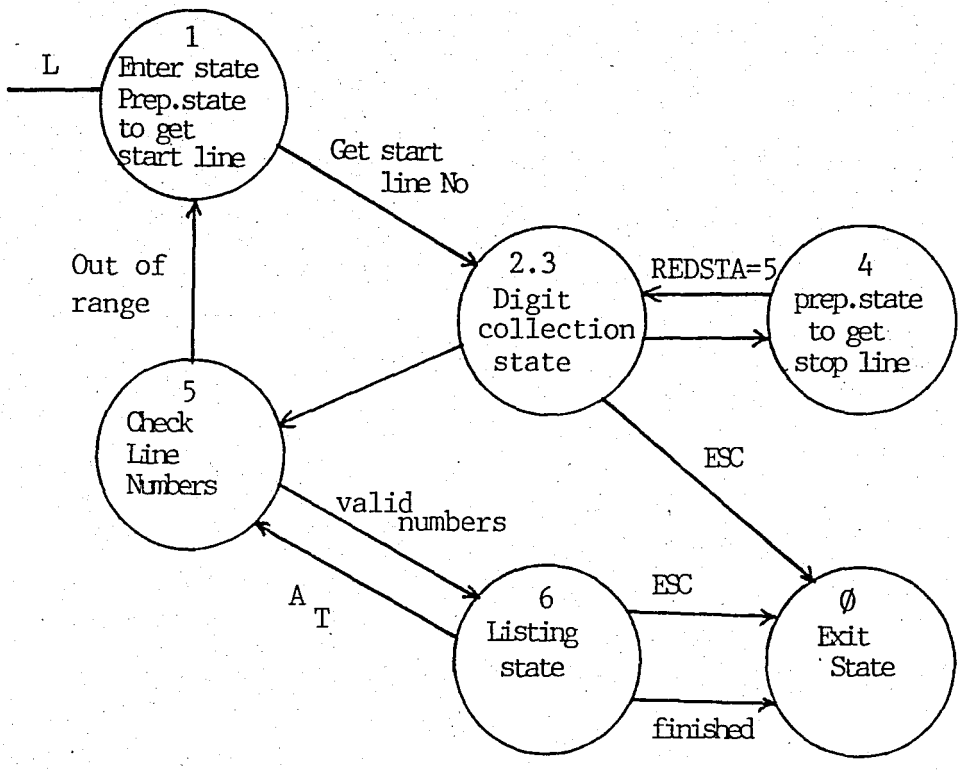


Fig.5.9 State diagram of LIST function

LSS000: Exit state from list routine.

LSS001: Initialization state for list command.

LSS002 and LSS003: The collection states of line number entries. In one activation of the list routine, the program can pass two time through these states, because this routine needs two line number.

LSS004: Preparation state for getting last line number.

LSS005: Control state of given first and last line number entries. If they are not valid (for example FIRST > LAST

etc.), the program ignores the given numbers, and returns to start state. If they are valid, the program goes on and jumps to display state.

LSS006: Displays the lines from given first line number to last line number. After each line displayed, the program waits a key from keyboard. If the pressed key is not a control key, the program goes on to display next line.

The control keys are "A" and "T".

"A"...to start display operation again from first line to last line previuosly defined.

"T"...to start display operation from the top of the memory to bottom.

The flow diagrams and the state diagram of the List routine are illustrated in Fig.5.9.

6. COMPILER

The most important module of UMSC S/W is the Compiler. It consists of two parts, one is Syntax Checker and the other is Interpreter. When "C" key is pressed by the user, if the system is on standby, The Master Control passes the control to the Compiler routine.

The first requirement of this module is the beginning line number which will be compiled area. The other requirement is the stop line number. This is done in a similar way which is used by the List routine. After giving those input data, the syntax checker part of the compiler begins to run. The flow

diagrams of this routine are illustrated in Fig.5.10 and Fig.5.11.

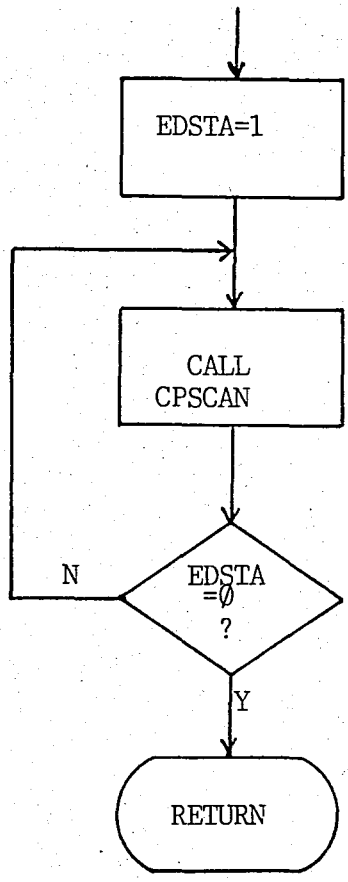


Fig.5.10 Flow diagram of CPTCMD routine

a. STATE DESRIPTIONS

CPS000: Exit state from compiler routine

CPS001: Initialization state for compilation

CPS002 and CPS003: First and Last number collection states. The compiler pass those states two times for two line number entries.

CPS004: Preparation state for getting last line number.

CPS005: Control state of given the first and the last line number entries. If they are not valid, the program ignores them and returns to start state (State 1). If they are valid, the program goes on and jumps to Syntax Checking state (State 6).

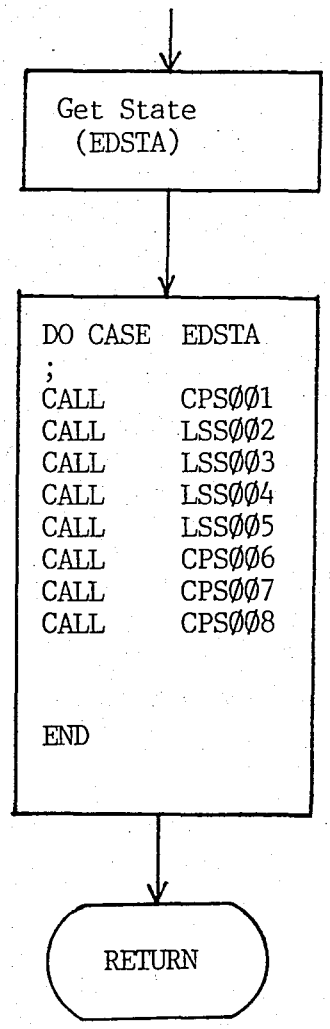


Fig.5.11 Flow diagram of CPSCAN routine

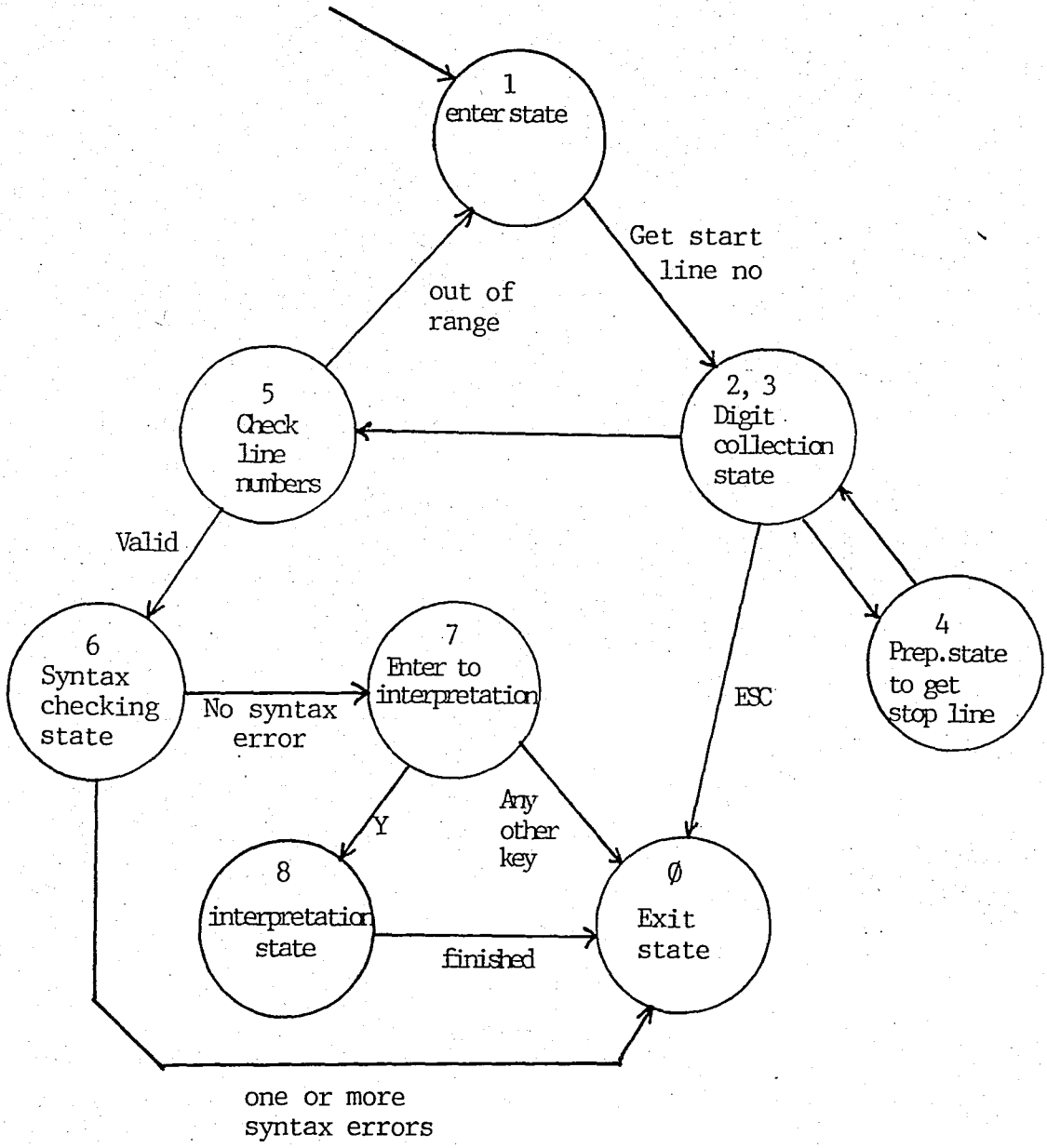


Fig.5.12 State diagram fo COMPILER routine

CPS006: Syntax Checking state, it takes each line beginning from first line to last line, checks them that they are empty or not. After skipping empty lines, it checks the

full lines. At the end of the operation if there is no error, according to decision of the user, it goes on to interpreter preparation state (State 7) or exits from compiler routine. If there are one or more errors, the program exits from compiler routine and enters to master control.

CPS007: User decision wait state to go on or exit.

CPS008: Interpretation state, It follows the same way with the CPS006 in order to skip empty lines. It takes each line and interpret it. After the interpretation of each line it checks the ENDFLG, If it is set, the interpretation is terminated and the control passed to the master control.

The state diagram of this routine is illustrated in Fig.5.12.

b. THE SYNTAX CHECKER

Firstly, it is necessary that the giving a command of the UMSC language from the line. Generally a command consists of four parts. They are; Instruction, operand, Data field (if exists), and command terminator.

INSTRUCTION: Consists of a character.

OPERAND: Consists of two characters. Operand characters might be <SP>, <Letter>, <Digit>.

DATA FIELD: There is not specified number of character in this field, but it is implicitly limited by the number of character in each line which is 32 bytes. The data field begins after the last operand character with a quotation mark and

finishes again with a quotation mark.

TERMINATOR: Consists of a character. It is a semicolon <;>.

The ASCII characters are used in those parts. A valid command does not exceed one line.

c. TYPES OF COMMANDS

There are eight different type commands as;

1.. <I><Digit,Digit><;>

2.. <I><Digit,Digit><"><Data Field><"><;>

3.. <I><Letter,Letter><;>

4.. <I><Letter,Letter><"><Data Field><"><;>

5.. <I><SP,Letter><;>

6.. <I><SP,Letter><"><Data Field><"><;>

7.. <I><SP,SP><;>

8.. <I><SP,SP><"><Data Field><"><;>

It is easily can be seen that there are four different type commands. The others are derived from them by adding a Data Field.

The compiler after getting start and stop line numbers, it activates the Syntax Checker. It takes a line and starts to search until a terminator is found, then it decides that is a command. During the terminator searching, it also moves the characters from line to a buffer. After then it calls the PARSING routine for the string placed into the buffer. This routine takes the first character which has to be an instruction

from the buffer and compares it with a look up table PARTBL. If one of item of PARTBL matches the first character in the buffer, the parsing goes on to define the type of instruction by looking following entry in the PARTBL. According to the type of command a parameter is passed to the another procedure which checks the buffer for operand and data field (if exists). This procedure is named CHK$TYP. By the result of type checking, the Syntax Checking operation of a command finishes. During those controls if any mismatching occurs, the program of syntax checker displays the type and location of error on the CRT monitor immediately. And also it increments an error counter. At the end of the last command checking, if the error counter is zero, the compiler asks to the user to go on to interpretation or to exit from compiler.

d. THE INTERPRETER

The interpreter is activated at state 7 and state 8, if it is desired by the user. It has a similar structure with the syntax checker. It takes instructions from a line one by one, and executes the program which is related with the instructions. After execution of each instruction the interpreter checks the keyboard. Because if any key was pressed, this is an interrupt for the interpreter. If it is so, The interpreter terminates the execution and passes the control to the master control. That is the way to exit from an infinite loop because of a logical error in the user program. The interpreter, after each instruction

also controls the JMP flag (JMPFLG), according to the status of JMPFLG, it goes on to next instruction or another line.

The interpreter calls the EXECUTIVE for each instruction. It takes a value from a look up table (EXECTBL) related with the taken instruction. And then the EXECUTIVE calls the instruction realization program by using the taken value.

7. GPIB ROUTINES

It can be discussed in four parts. These are Initialization, Remote Enable, Send and Receive routines.

a. INITIALIZATION

This routine initialize 8291A (GPIB Talker/Listener) and 8292 (GPIB Controller) chips. Therefore the GPIB is initialized.

The only action for initialization of 8292 is to enable the TCI (Task Completed Interrupt) interrupt mask.

The initialization of 8291A is;

-to disable both major and minor addresses. Because 8291A will never see the 8292's commands.

-to set Talk only mode

-to set internal counter to match to the clock input of 8291A.

-to clear all interrupts of 8291A.

The flow diagram of this routine is shown in Fig.5.13.

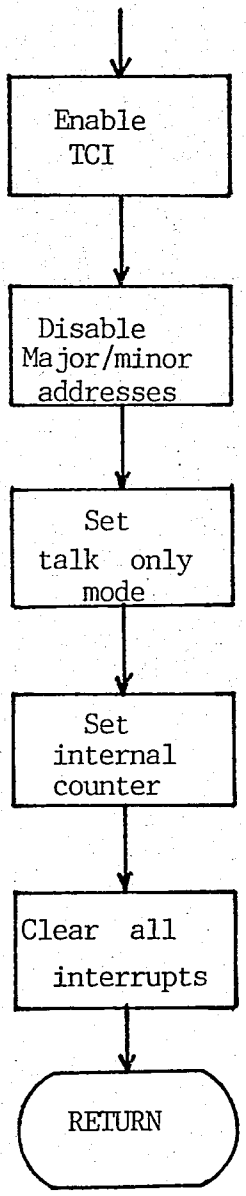


Fig.5.13 Flow chart of GPIB initialization

b. REMOTE ENABLE

This is done by using a feature of 8292. A special code which is written into an input register of 8292 provides it.

The flow diagram of this routine is shown in Fig.5.14.

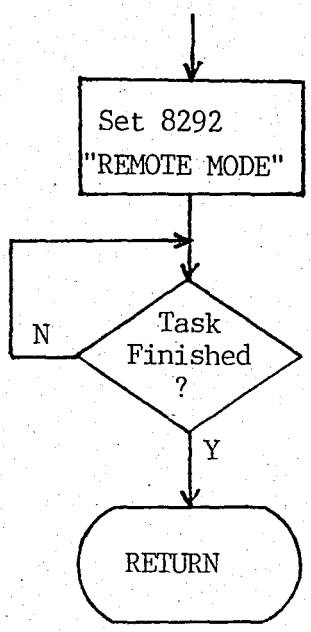


Fig.5.14 Flow chart of Remote enable

c. SEND

First all devices on the bus are put Unlisten mode, and then desired device is addressed by using its device address to listen. After that 8291A is enabled to send data bytes, and the 8292 is put to Standby. Now 8291A can start to send data bytes to addressed device via GPIB. At the transmission of last byte, EOI line is activated to imply that this byte is the last one. After that, 8292 takes the control of bus again. The flow diagram of this routine is illustrated in Fig.5.15.

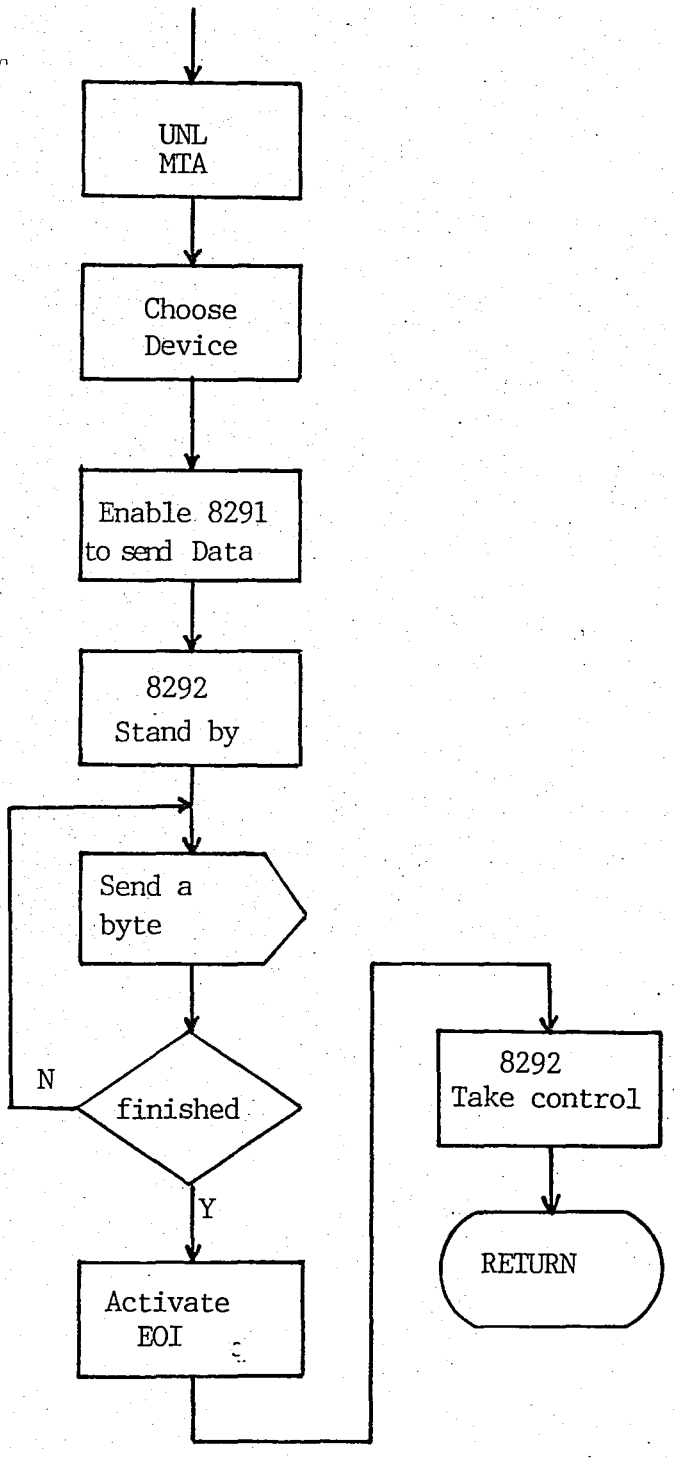


Fig.5.15 Flow chart of Send routine

d. RECEIVE

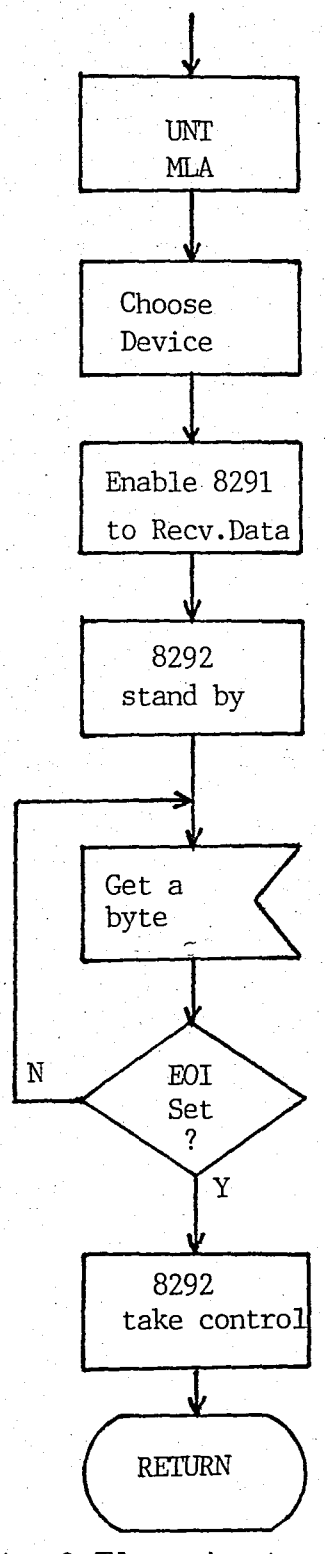


Fig.5.16 Flow chart of Receive routine

This routine is the dual of the send routine. First all devices on the bus are put on Untalk mode, and desired device is addressed by using its device address to talk. After that 8291A is enabled to receive incoming bytes from GPIB and 8292 is put to Standby. And then 8291A begins to receive data bytes from bus. After receiving of the last byte which is pointed by EOI line, 8292 takes the control of GPIB. The flow diagram of this routine is illustrated in Fig.5.16.

VI. MAN MACHINE INTERFACE

A. INTRODUCTION

The UMSC can work in four different modes. Each of these modes can be activated by special key while the controller in SYSTEM MODE.

B. SYSTEM MODE

When the controller is power on or reset, It will enter to the system mode and displays a message and a prompt (#);

```
GPIB SYSTEM IS READY
#
```

While the system is in this mode the <#> prompt is on the CRT. After the realization of each task this prompt is displayed. If any wrong key is pressed, an error message is displayed;

```
SYSTEM ERROR !
#
```

The system mode commands are activated by using upper case letters. Those are;

<E>.... Enters to the EDITOR

<L>.... Enters to the LIST routine

<C>.... Enters to the COMPILER

<M>.... Enters to the MONITOR

C. EDITOR MODE

In this mode, the controller edits user programs and stores it into a nonvolatile memory. The editor of UMSC is a line editor. The nonvolatile memory is divided into 60 lines. Each line is 32 characters of length. The user can copy a line, delete a line, delete previous character, clear user program area. The lines are sequential in the memory. So the lines are automaticaly numbered. The user can begin to edit from any line.

Enter to Editor:

While the controller is in the SYSTEM MODE, by pressing <E>. a message is displayed;

```
LINE EDITOR
nnn LINE(S) FREE
TYPE LINE NUMBER
$
```

If all lines are used, a full area message is given;

```
NO IDLE LINE
PRESS RUBOUT KEY TO CLEAR ALL LINES
PRESS SPACE TO RETURN LINE EDITOR
PRESS ANY KEY TO EXIT
$
```

Entering Line Number:

After "TYPE LINE NUMBER" message is displayed, the user can type line number. The controller only accepts digits. The

line number can be between 0 and 59.

Editing a line:

The editor displays the desired line number with its content in the first line, and opens a new one with the given line number. After that, the user can edit line by using keyboard. For example;

```
 001 O23"F4,R5";
$001
```

or

```
 001
$001
```

Copying a character:

It can be made by using cursor right key, if the line was previously edited. By this way a wrong character can be changed. For example; Changing R5 to R6

```
 001 O23"F4,R5";
$001 O23"F4,R6";
```

Changing previous charater:

It can be done by using cursor left or back space. The cursor is moved one character left and new character is written.

New line:

The carriage return is used to finish the current line and continue the next line.

Delete a line:

This is done by using RUBOUT. The editor deletes the current line and passes the next line.

End of a line:

Each line can have 32 characters, so while edition of last two character, the user is warned by a "BEEP" sound. After the edition of last charater, the editor passes to next line.

Exit from Editor:

By pressing <ESC> key, the user can exit from the Editor. If one or more character is written, the editor saves them into the nonvolatile memory and the exits. If no character is written into the line or while enterring the line number, the editor just exits and returns to the System Mode.

Note 1: After the edition of last line (line 59), a message is displayed;

```
NO MORE IDLE LINE
PRESS SPACE TO RETURN LINE EDITOR
PRESS ANY KEY TO EXIT
$
```

Note 2: When enterred to editor if all lines were used,

a message is displayed:

```
NO IDLE LINE
PRESS RUBOUT KEY TO CLEAR ALL LINES
PRESS SPACE TO RETURN LINE EDITOR
PRESS ANY KEY TO EXIT
$
```

If the rubout key is pressed, a message is displayed;

WAIT

After about 20 sec another message is displayed;

```
PRESS SPACE TO RETURN LINE EDITOR
PRESS ANY KEY TO EXIT
$
```

D. LIST FUNCTION

The format of List command is as below;

LIST FROM nn TO mm

Where the nn is beginning line number and mm is stop line number. For example listing of program written between line 0 to 3.

LIST FROM 00 TO 03

```
000 O23"F4,R5";W20;
001 I23"W";
002 D W;
003 E  ;
```

To list the program any key have to be pressed continuosly

To stop the list no key is pressed

To list from given first line again, <A> key is pressed

To list from top of the EEPROM area, <T> key is pressed

To exit , <ESC> key is pressed.

```
LIST FROM 07 TO 15

007 O23"F4,R5";
008 W20;
009 I23"W";
010 D W;                    If <A> is pressed

007 O23:F4,R5";
008 W20;                    If <T> is pressed

000 O10"S3,T5";
001 W50;                    I f <ESC> is pressed

#
```

E. COMPILER

The compiler is consists of two parts; The Syntax Checker and The Interpreter. The user can enter to the compiler via the syntax checker. After pressing <C> while controller is ni System Mode, the compiler is active.

The format of compilation start command is as below;

COMPILE FROM nn TO mm

Where the nn is start line number and mm is stop line

number.

After giving those numbers, a message is displayed on the CRT monitor.

RUNNING

While the syntax checker is active, all the syntax errors are displayed on to the CRT monitor.The error messages will be defined later.

The syntax checker calculates the number of errors and displays it at the end of the syntax checking process. If at least one error exists, the compilation is terminated and the controller returns to the System mode.

If there is no error, the compiler messages to the user;

```
NO ERROR(S) FOUND
PRESS <Y> TO EXECUTE THE PROGRAM
PRESS ANY KEY TO EXIT
```

If <Y> is pressed, the compiler passes to the second part(Interpreter), and begins to interpretation of user program.

If any other key is pressed, the controller enters to the System Mode.

F. MONITOR

This routine can be activated by pressing <M>, when the

controller is in System mode and terminated by pressing <ESC>, when the controller is in Monitor mode. This routine is for manupilating the controller hardware and data structures manually. The instructions are defined as below;

1. MONITOR INSRUCTION SET

SUBSTITUTE COMMAND: (>Sabcd)

When addressing the entry last 4 digits are considered to be valid, also less than 4 digits may be entered. When space bar is pressed current entry is displayed.

>S2379 46-FF 46

Subsitute operation is performed only after a space bar or carriage return. Prompt is displayed following a carriage return.

DISPLAY COMMAND: (>Dabcd-uvwy)

abcd gives start address (last 4 digits) and uvwy gives end address (always last 4 digits, also less than 4 digits can be used). After carriage return, the required memory block is displayed in the format shown below:

```
>D0 20
0000 C3 40 00 FF FF FF FF FF FF FF FF FF FF FF FF FF
0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020 FF
```

To stop display operation ESC key can be used.

INPUT COMMAND: (>I pq xy)

Performs input operation from port designated by pq and displays data read as xy eg.

>I 60 02

means input operation from port 60 reads 02 . If after port number carriage return is pressed the data is displayed as follows:

```
>I 60
02
>
```

OUTPUT COMMAND: (>O pq xy)

Performs output operation of data xy from port pq. Operation is performed after pressing space bar or carriage return.

>O 40 30

DISPLAY REGISTER COMMAND: (>X)

Displays the register values to be used when G,T or B command is to be used. The sequence is as follaws:

```
XF=00 A=01 C=02 B=03 E=04 D=05 L=06 H=07 S=2700 P=3800 #=3807

F...flags
A...Accumlator
C...C register
B...B register
E...E register
```

D...D register
L...L register
H...H register
S...Stack pointer
P...Program counter
#...Break point address.

Any of these values can be changed as follows: eg.

XC=02-73 B=03-

After enterring 73, pressing carriage return also performs the required change operation. This time prompt is displayed instead of next register.

GO COMMAND: (>G)

Starts executing the code located at the memory pointed by program counter. eg. if program counter has the value 3800H, G command causes control to be passed to the program starting at location 3800H. Only after a return instruction outside nested CALL's and RETURN's control returns to monitor. Stack pointer should be initialized to a value such that called program can use an unused portion of RAM as stack.

GO COMMAND WITH BREAK POINT: (>B)

Before performing this command, program counter, stack pointer and break point values should be entered properly. Break point sholud be the address of the memory location containing the first operand of an assembler instruction. After every instruction control is returned to the monitor. Therefore programs executed with breakpoint require more time than executed

with go command.

SINGLE STEP: (>T)

Before performing this command, program counter and stack pointer should be updated. Any valid assembler instruction can be executed. If instruction does not destroy workspace area of monitor, or disturb the communication hardware interfacing to the user control returns to the monitor after every instruction (only for HLT instruction this is not possible).

G. UMSC LANGUAGE

Variables: There are two kinds of variables; integer and real.

K and L integer are integer variables. They can be used for counter.

U,V,W,X,Y,Z are real variables. Each of them is consists of 32 bytes length. The value are stored in ASCII form in them. They can be displayed.

1. INSTRUCTIONS OF UMSC LANGUAGE

All instructions are single character instruction. Some of them needs operand and data.

OUTPUT:

This instruction performs output a character string to

the GPIB.The syntax of instruction is as follow;

O<operand><data field>

Where operand is two digit which is specifies the number of device will receive the character string. In the IEEE 488 standard device numbers should be between 0 and 31. The data field is the character string.

O24"F4,R5,W0,?";

the commas are ignored in the data field.

INPUT:

This instruction performs input a character string from GPIB which is sent device specified by operand. The syntax of instruction is as follow;

I<operand><data field>

Where the operand represents the device address.Similarly it should be between 0 and 31 Data field is point the which register will used to store input data.

I24"X";

WAIT:

This instruction performs a wait operation before the execution of the next instruction. The unit delay is 250msec. The syntax of instruction is as follow;

W<operand>;

Where the operand is specifies the number of unit delay. It can be between 0-99. Therefore maximum delay is 25sec.

EQUAL:

This is for initializing the K or L register. The syntax of instruction is as follow;

=<operand><data field>

Where the operand is specifies K or L register which will be inialized by the value specified in data field. It should be a space between instruction character and register name.

= K"230";

DECREMENT/INCREMENT:

This instruction decrements/increments the value of integer variable. The syntax of instruction is as follows;

-<operand> / +<operand>

Where operands specify K or L variables with a space character between instruction character and integer variable name. The real variable names can not be used.

- K;

or

+ L;

COMPARE:

This is for comparing the values of two integer variables. The syntax of instruction is as follow;

C<operand>

Where operand specifies the sequence of variables in the

comparision. This instruction can be performed only integer variables.

CKL; means K > L ?

or

CLK; means L > K ?

At the result of comparision, if the condition is satisfied the next instruction will be executed. If it is not so, next line will be executed.

JUMP:

This instruction jumps the soecified line and goes on from that line. The syntax of instruction is as follows;

J<operand>

Where the operand points the line number. It should be in the area which is specified by the start and stop line numbers given at the beginning of compilation. And also it should be between 0 to 59. Because there is 60 lines specified sequencially for writting user program

J23;

S COMMAND:

This is a special instruction for giving a decision to user. It terminates the execution of current user program. And then user can give decision of going on the part of program or restarting from beginning line of program or exit from compiler and return to system mode. The syntax of instruction

is as follows;

S<operand>

Where the operand consists of two space character and is not important.

S ;

During the compilation if the current instruction is a S command, a message is displayed, that is;

```
PRESS <C> TO CONTINUE
PRESS <R> TO RESTART
PRESS <E> TO EXIT
```

This instruction can be used for debugging a part of user program or to give a delay to user for changing measured resistor or capacitor e.g. with new one.

REMARK:

This is for giving information about the program. It is not executed. The syntax of instruction is as follows;

R<operand><data field>

Where the operand is two space charater and it is not important. The data field consists of the given information.

R "RESISTOR TEST PROGRAM";

END:

This instruction shows the end of program. It should be all the written programs. The compiler terminates the execution

of the user program with this instruction. The syntax of instruction is as follow;

E<operand>

Where the operand is two space character.

E ;

> COMMAND:

This instruction displays the GPIB Input buffer on to the CRT monitor. It can be used after a INPUT (I) instruction, to see incoming data. The syntax of instruction is as follow;

><operand>

Where the operand is two space characters.

I23"X";> ;

D and B COMMANDS:

These instructions are for the displaying contents of one of the six real register with/without a comment on the CRT monitor. One difference between D and B commands is that warning the user with a BEEP sound while execution of B command. So the user can trace the results and can interrupt the execution of current user program. The syntax of instructions are as follow;

D<operand><data field>

or

B<operand><data field>

Where the operands can be a space and a letter which

specifies one of the six real variables.The data field consists of comments.

D V"Measured Value= ";

or

B X"Frequency = ";

If the value of V is 23.34 E03 KOHM, the effect is on the CRT monitor is as follow;

Measured Value= 23.34 E03 KOHM

P COMMAND:

This instruction is similar to D and B commands except that it sends the results and comments to the printer. The syntax is similar to D and B commands.

P<operand><data field>

P W"Value of resistor is ";

If the value of W register is 12.929 OHM, it can be seen on the print out as follow;

Value of resistor is 12.929 OHM

H. ERROR MESSAGES

While the syntax checking, if there is a writting fault in the instruction or line, the compiler sends an error message which specifies the type of error and points line number and location of error in the line.

SYNTAX ERROR:

If the parts of instruction are confused, it causes a Syntax error. e.g.

002 I24"X"; W20;

A space, after the first instruction causes this type error such that;

SYNTAX ERROR IN 002 NEAR W20

UNRESOLVED INSTRUCTION ERROR:

If an unexistent instruction character is used, it causes an unresolved instruction error. e.g.

002 Z00;

This causes as follow error message;

UNRESOLVED INSTRUCTION IN 002 NEAR Z00

WRONG OPERAND:

If the operand of instruction is undefined type, it causes a Wrong operand error. e.g.

003 W0A;B03"value =";

This line of code causes two wrong operand error messages.

WRONG OPERAND IN 003 NEAR W0A
WRONG OPERAND IN 003 NEAR B03"value ="

UNEXISTENT VARIABLE:

If there is unexixtent varible in operand or data field, it causes an Unexistent variable error e.g.

```
003 + A;I23"Q";B H"value is ";
```

This line of code causes three wrong unexistent variable error message.

```
UNEXISTENT VARIABLE IN 003 NEAR + A
UNEXISTENT VARIABLE IN 003 NEAR I23"Q"
UNEXISTENT VARIABLE IN 003 NEAR B H"value is"
```

UNTERMINATED INSTRUCTION:

All the instruction should be terminated by <;>. If it is not so, an unterminated instruction error message occurs on the CRT monitor. e.g.

```
003 I20"W";R  "INPUT VALUE"
```

This line of code causes an error message as follow;

```
UNTERMINATED INSTRUCTION IN 003 NEAR R  "INPUT VALUE"
```

MISSING QUOTATION:

If the quotations which specify the data field are missing, a missing quotation message occurs on the CRT monitor. e.g.

```
002 O20F4,R3,S6";
```

```
003 I20"W;
```

This program part causes two error messages as follows;

```
MISSING QUOTATION IN 002 NEAR O20F4,R3,S6"
MISSING QUOTATION IN 003 NEAR I20"W
```

INVALID CHARACTER:

If there are more character than the desired quantity, it causes an invalid character error on the CRT monitor. e.g.

```
002 I20"W A";B W"MEASURED VALUE =";
```

This line of code causes an error message as follow;

```
INVALID CHARACTER IN 002 NEAR I20"W A"
```

END NOT FOUND:

All the user programs should be lasted by an END instruction, if it is not so, an End not found error occurs on the CRT Monitor. e.g.

```
000 R  "RESISTOR MEASURING";
001 O20"F4,R3,S5,?";W35;
002 I20"W";>  ;
003 P W"RESISTOR VALUE = ";
```

If the compilation was begun by giving 00 as start line number and 03 as stop line number, At the end of the compilation an end not found error message is given such that;

```
END NOT FOUND IN 003 NEAR
```

OUT OF RANGE:

This kind error occurs when the operand are digits and also they are out of range. e.g.

```
000 O20"F4,R5,S5,?";
001 = K"100";= L"0";I45"W";
002 + L;CLK;J45;
003 D W"VALUE";
004 J01;
005 E  ;
```

This user program causes error as follows;

```
OUT OF RANGE ERROR IN 001 NEAR I45"W"
OUT OF RANGE ERROR IN 002 NEAR J45
```

WRONG DATA TYPE:

If there are letters or characters different from digits, while the compiler waiting digits in data field. This kind of error occurs on the CRT monitor. e.g.

```
003 = K"AD";
```

This instruction causes a wrong data error, such that;

```
WRONG DATA IN 003 NEAR = K"AD"
```

VII. CONCLUSION

In the previous sections of this thesis, the GPIB was defined, the hardware and software structure of UMSC were described in detailed form.

With the design of UMSC, a basic language was created. It has 15 different instructions such as jump, compare and increment/decrement. Therefore the basic software routines can be written by using UMSC language. In future, some new instructions can be added easily. The structure is available to do it. The UMSC language is not one of the well known high level languages. So some features of them can not be found in it such as floating point arithmetic, arrays etc.. However they can be implemented by changing the hardware and software structure of the system in future, if desired.

The line editor of UMSC was designed to answer most of the edition problems of user. The finest solution is a screen editor. Different types of CRT monitors use different control characters for formatting the screen. One of the design aim of UMSC is compatibility for all CRT monitors. Therefore a screen editor could not be realized in UMSC.

The compiler was realized in the two pass form. The first pass checks syntax of instructions and displays errors on CRT monitor. If there are no errors in the first pass, the compiler allows the second pass. It interprets and executes the

user program.

With the design of the UMSC, different type of measurement instruments which have remote control capability, can be controlled easily via the GPIB. The UMSC offers the possibility of a printer to take print-out of measurement results. It also offers a parallel input/output port different from GPIB for future developments such as communication with another microcomputer or to drive a relay circuit which is established for the test set.

In the current design of UMSC, it is provided that 2x8K EPROM, 2x2K EEPROM, and 2x2K RAM memory comfiguration is available. It can be increased up to 64K totally for future applications.

If the user has a line printer which can be controlled by GPIB, the serial port designed for communication with printer can be changed for communication with another system by minimum S/W and H/W changing.

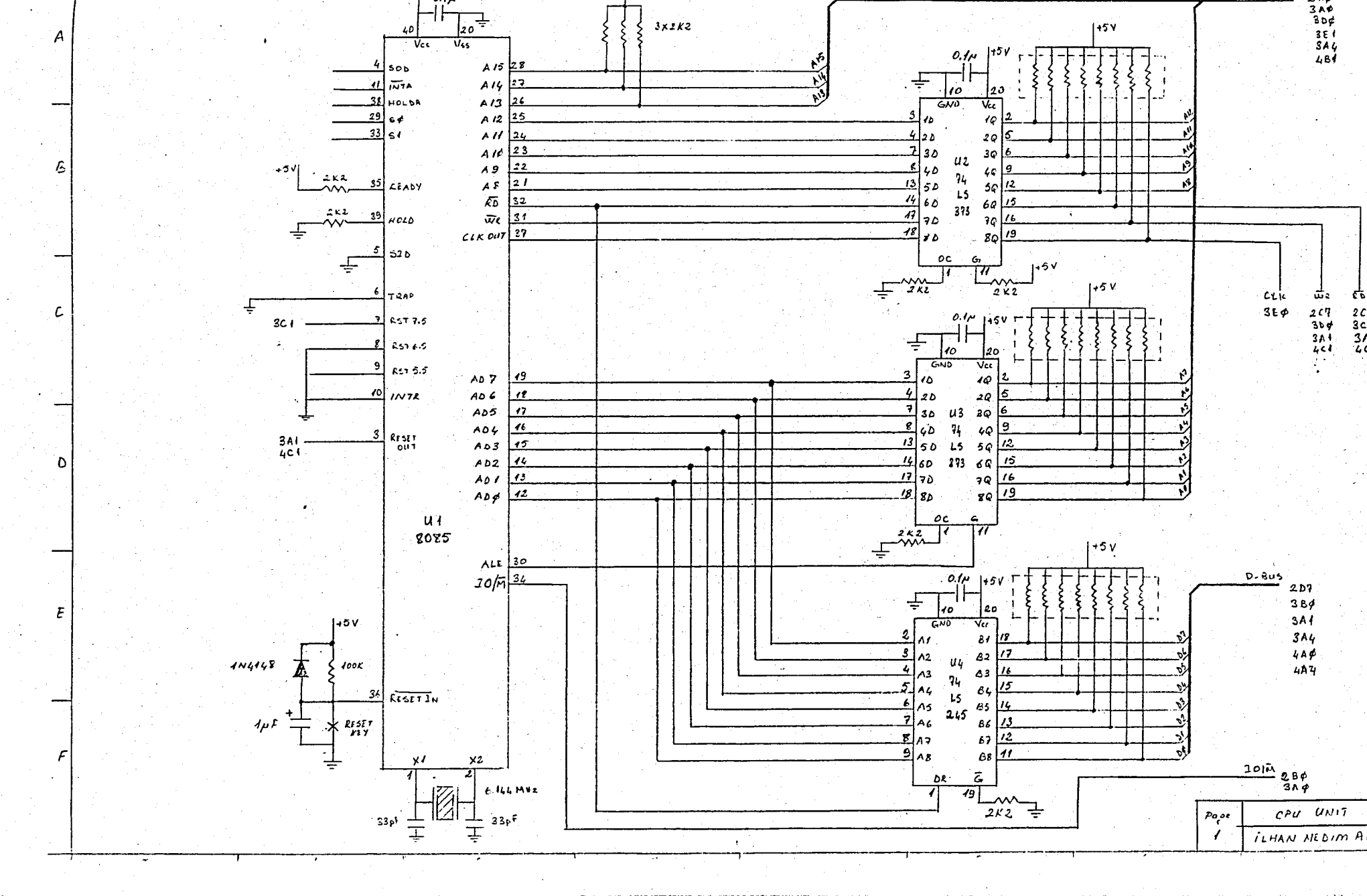
It is designed for using individually, but it can be changed easily to GPIB module for a personal computer by establishing necessary parallal or serial communication between them.

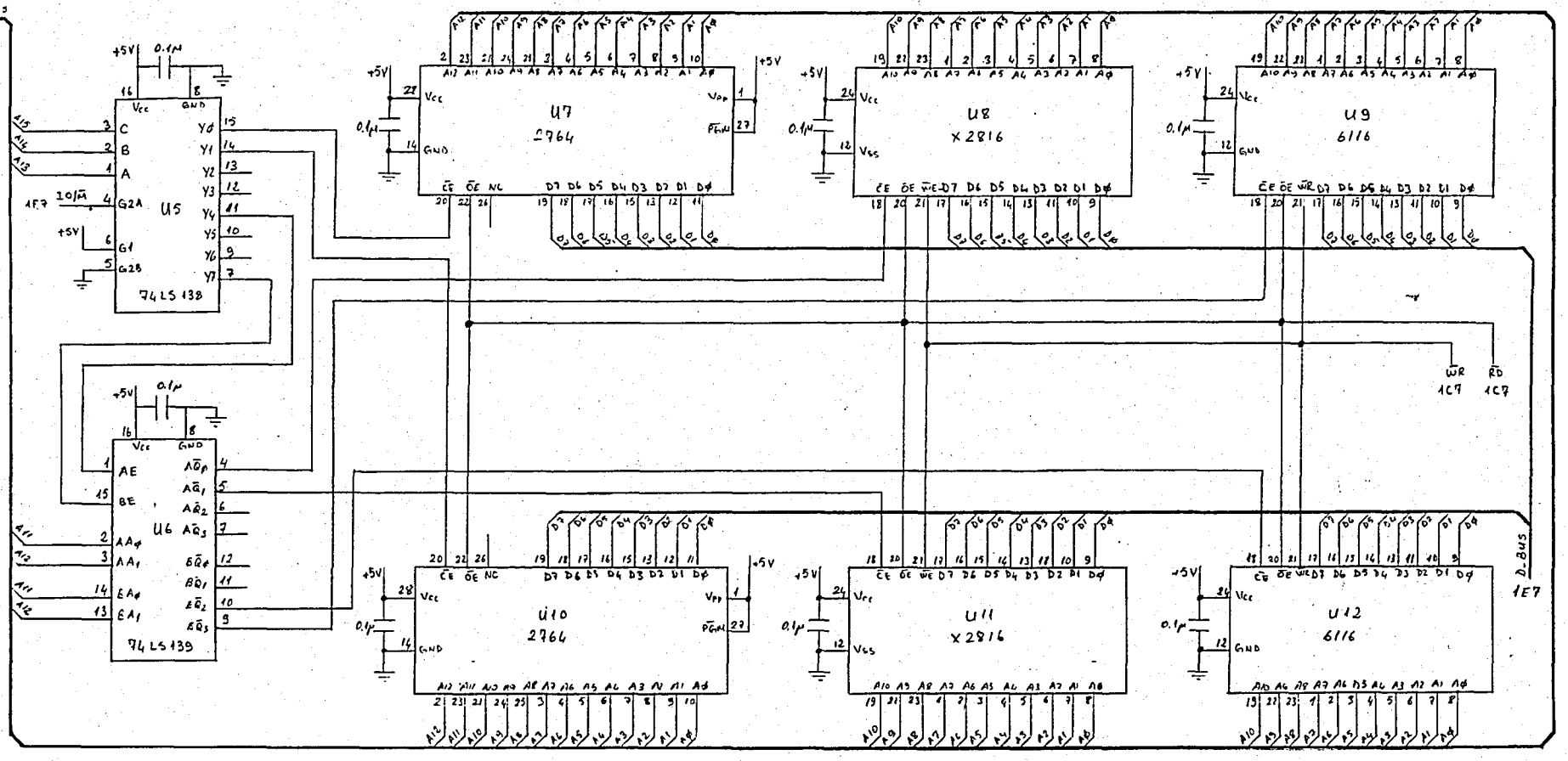
For the future developments in UMSC, a CRT controller and a CRT monitor with a keyboard can be added to obtain a compact system. In order to increase memory capacity a floppy disk drive can be applied. It might be necessary a DMA controller to handle memory operations. Another development can

be done in the way of changing UMSC as GPIB board for microcomputers. UMSC can act upon the microcomputer as a port and works according to the task given by the microcomputer. By using this approach microcomputer is free from consuming time for GPIB communication control process.

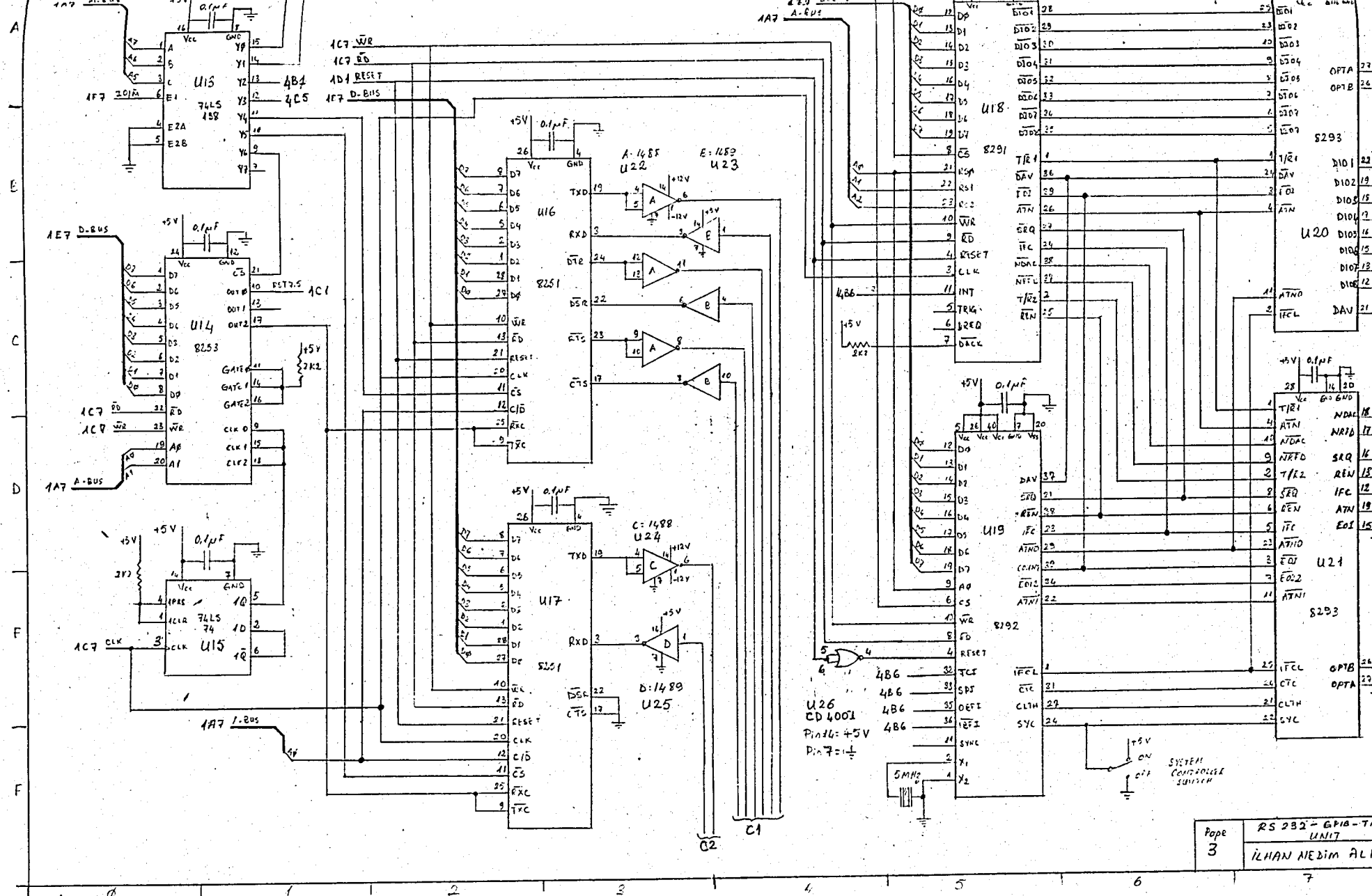
APPENDIX A

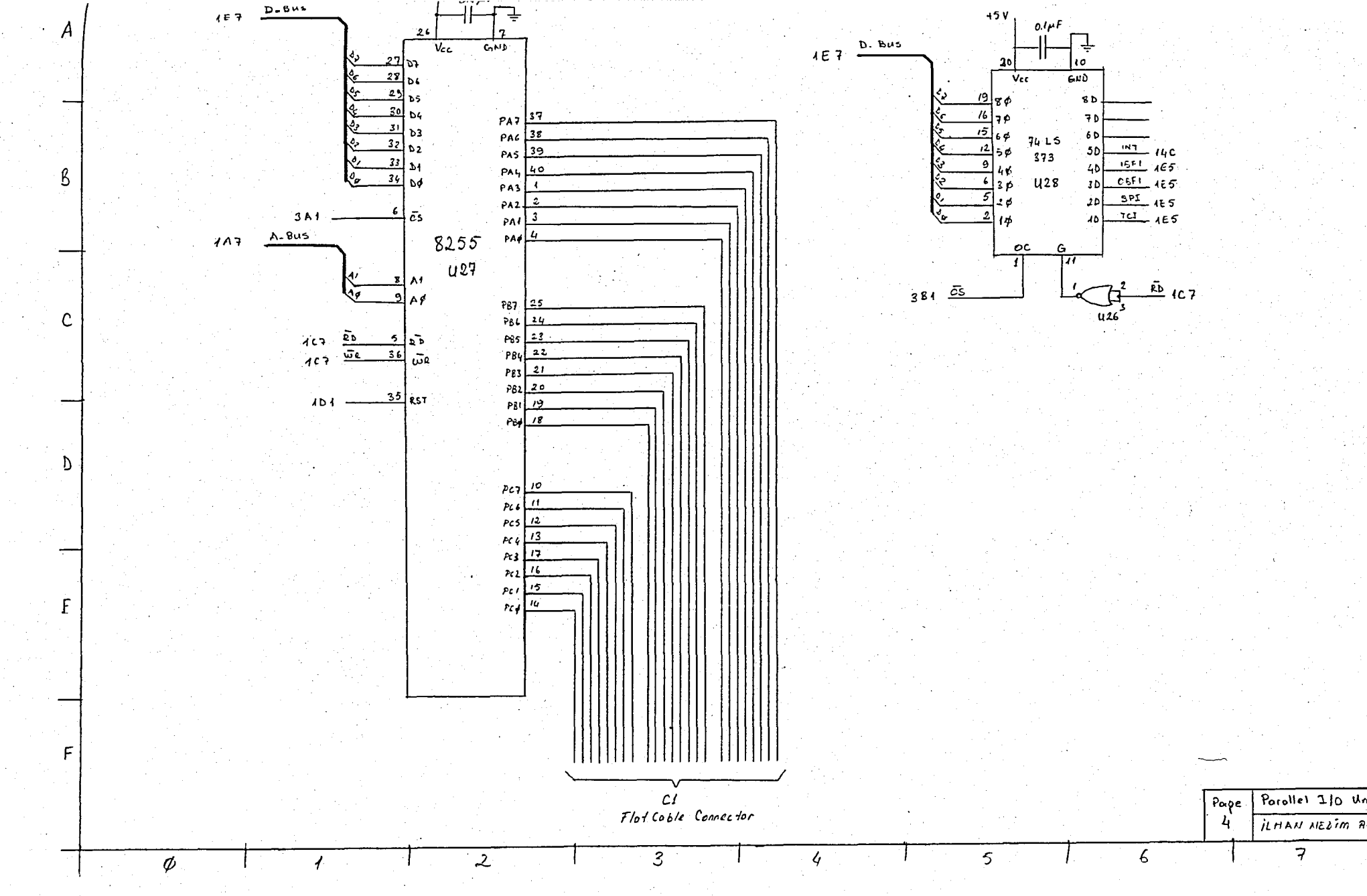
CIRCUIT SHEMATICS


U1
8085
U2
74 LS 373
U3
74 LS 373
U4
74 LS 245
X1
X2
6.144 MHz
33pF
33pF
1µF
100K
1N4148
RESET IN
RESET OUT
SOD
INTA
HOLDA
IO/M
SO
SID
TRAP
RST 7.5
RST 6.5
RST 5.5
INTR
READY
HOLD
ALE
RD
WR
CLK OUT
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
A15
A14
A13
A12
A11
A10
A9
A8
3X2K2
2K2
0.1µ
+5V
D-BUS
Page 1
CPU UNIT
İLHAN NEDİM A



Page	Memory Unit
2	İLHAN NEDİM A


U15
74LS 138
U16
8251
U17
8251
U14
8253
U18
8291
U19
8292
U20
8293
U21
8293
U22
U23
U24
U25
U26
CD 4001
Pin14: +5V
Pin7: ⏚
5MHz
SYSTEM CONTROLLER SWITCH
C1
C2
RS 232 – GPIB – T
UNIT
İLHAN NEDim AL
Page 3



Page	Parallel I/O Un
4	İLHAN NEZİM A

APPENDIX B

SOURCE LISTINGS

```
$PAGEWIDTH(80)
;*************************************************
;*                                               *
;*        UNIVERSAL MEASUREMENT SYSTEM           *
;*            CONTROLLER WITH GPIB               *
;*                                               *
;*                INITIALIZATION                 *
;*                                               *
;*  FILE NAME   : IN001A.S03                     *
;*  MODULE DEF. : VECTOR TABLE                   *
;*  DATE        : MARCH 09 1986                  *
;*  AUTHOR      : ILHAN NEDIM ALP                *
;*                                               *
;*************************************************
;
        NAME    INIVEC
;
        PUBLIC  VECTOR
;
        EXTRN   INPORT
        EXTRN   RST75
;
        CSEG
;
;*************************************************
; PROCEDURE : VECTOR
; FUNCTION  : Controls the system addressing during
;             interrupts, reset and TRAP
; PLM CALL  : NONE
; INPUT     : The last reset or trap address
; OUTPUT    : NONE
; GLOBALS   : NONE
; CALLS     : INPORT
; DESTROYS  : All
;
VECTOR:
        JMP     INPORT  ;RST 0
        RST     0
        RST     0
        RST     0
        RST     0
        RST     0
        JMP     INPORT  ;RST 1
        RST     0
        RST     0
        RST     0
        RST     0
        RST     0
        JMP     INPORT  ;RST 2
        RST     0
        RST     0
        RST     0
        RST     0
        RST     0
        JMP     INPORT  ;RST 3
        RST     0
        RST     0
        RST     0
        RST     0
        RST     0
```

```
        JMP     INPORT  ;RST 4
        RST     0
        JMP     INPORT  ;TRAP
        RST     0
        JMP     INPORT  ;RST 5
        RST     0
        JMP     INPORT  ;RST 5.5
        RST     0
        JMP     INPORT  ;RST 6
        RST     0
        JMP     INPORT  ;RST 6.5
        RST     0
        JMP     INPORT  ;RST 7
        RST     0
        JMP     RST75   ;RST 7.5
        RST     0
;
        END
;
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*         UNIVERSAL MEASUREMENT SYSTEM          **
;*             CONTROLLER WITH GPIB              **
;*               INITIALIZATION                  **
;*                                               **
;*  FILE NAME   : IN002A.S02                     **
;*  MODULE DEF. : INIT. OF 8255 AND OFF-LINE     **
;*  DATE        : NOV. 30 1985                   **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************

;
        NAME    INIPOR
;
        PUBLIC  INPORT
;
        EXTRN   INIALL
        EXTRN   PORTCM
;
        CSEG
;
;*************************************************
;  PROCEDURE : INPORT
;  FUNCTION  : INITIALIZATION OF 8255
;  PLM CALL  : NONE
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : PORT's
;  CALLS     : NONE
;  DESTROYS  : A
;
INPORT:
;
; Initialization of  8255 of the System
; A port OUT, B port OUT, C port IN/OUT
;
        DI
        MVI     A,88H ; control word
        OUT     LOW PORTCM
        NOP
        NOP
        NOP
;
        JMP     INIALL
;
        END
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*         UNIVERSAL MEASUREMENT SYSTEM          **
;*              CONTROLLER WITH GPIB             **
;*                                               **
;*                 INITIALIZATION                **
;*                                               **
;*  FILE NAME   : IN003A.S07                     **
;*  MODULE DEF. : INITIALIZATION OF ALL SYSTEM   **
;*  DATE        : JULY 31 1986                   **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************

;
        NAME    ALLINI
;
        PUBLIC  INIALL
;
        EXTRN   INIHW
        EXTRN   MASTER
        EXTRN   STKEND
        EXTRN   INZERO
        EXTRN   INIT
        EXTRN   REME
;
        CSEG
;
;*************************************************
;  PROCEDURE : INIALL
;  FUNCTION  : INITIALIZES ALL OF THE SYSTEM
;  PLM CALL  : NONE
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : INIHW, INZERO, INIT, REME
;  DESTROYS  : ALL
;
INIALL:
        MVI     A,1FH   ; mask of SIM
        SIM             ; set interrupt mask
        LXI     SP,STKEND; init stack pointer
        CALL    INIHW   ; initilization of hardware
        CALL    INZERO  ; initilization of data area
        CALL    INIT    ; initilization of GPIB chips
        CALL    REME    ; put GPIB bus into remote mode
;
        MVI     A,1BH   ; enable RST 7.5
        SIM
;
        EI
;
        JMP     MASTER
;
        END
;
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               *
;*         UNIVERSAL MEASUREMENT SYSTEM          *
;*             CONTROLLER WITH GPIB              *
;*                                               *
;*                INITIALIZATION                 *
;*                                               *
;*  FILE NAME   : IN004A.S03                     *
;*  MODULE DEF. : HARDWARE INITIALIZATION        *
;*  DATE        : MARCH 09 1986                  *
;*  AUTHOR      : ILHAN NEDIM ALP                *
;*                                               *
;*************************************************
;
        NAME    INICPU
;
        PUBLIC  INIHW
;
        EXTRN   TIMER0,TIMER1
        EXTRN   TIMER2,TIMERC
        EXTRN   USRTPC,USRTMC
;
; mode & command definitions
;
UMODEM  EQU     4FH ; Usart mode & commands
UMODEP  EQU     4FH
UCOMM   EQU     25H
UCOMP   EQU     25H
;
        CSEG
;
;*************************************************
;  PROCEDURE : INIHW
;  FUNCTION  : INITIALIZATION OF THE CPU HW
;  PLM CALL  : NONE
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : ADDRESSes OF USARTs,TIMERs
;  CALLS     : NONE
;  DESTROYS  : A
;
;
INIHW:
;
; Initialization of  8253 of the System
;
        MVI     A,36H ; control word for counter 0
        OUT     LOW TIMERC
        NOP
        MVI     A,80H ; least sign. byte of count 0
        OUT     LOW TIMER0
        NOP
        MVI     A,07H ; most sign. byte of count 0
        OUT     LOW TIMER0
        NOP
;
        MVI     A,76H ; control word for counter 1
        OUT     LOW TIMERC
        NOP
```

```
        MVI     A,00H ; least sign. byte of count 1
        OUT     LOW TIMER1
        NOP
        MVI     A,3CH ; most sign. byte of count 1
        OUT     LOW TIMER1
        NOP
;
        MVI     A,0B6H ; control word for counter 2
        OUT     LOW TIMERC
        NOP
        MVI     A,14H ; least sign. byte of count 2
        OUT     LOW TIMER2
        NOP
        MVI     A,00H ; most sign. byte of count 2
        OUT     LOW TIMER2
        NOP
;
; initialization of the usarts
;
; Monitor's USART
;
        XRA     A
        OUT     LOW USRTMC   ; sync
        OUT     LOW USRTMC
        OUT     LOW USRTMC
        MVI     A,40H    ; reset 8251
        OUT     LOW USRTMC
        NOP
        MVI     A,UMODEM
        OUT     LOW USRTMC
        NOP
        MVI     A,UCOMM
        OUT     LOW USRTMC
        NOP
;
; Printer's USART
;
        XRA     A
        OUT     LOW USRTPC   ; sync
        OUT     LOW USRTPC
        OUT     LOW USRTPC
        MVI     A,40H    ; reset 8251
        OUT     LOW USRTPC
        NOP
        MVI     A,UMODEP
        OUT     LOW USRTPC
        NOP
        MVI     A,UCOMP
        OUT     LOW USRTPC
        NOP
;
        RET
;
        END
;
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*         UNIVERSAL MEASUREMENT SYSTEM          **
;*             CONTROLLER WITH GPIB              **
;*                INITIALIZATION                 **
;*                                               **
;*  FILE NAME   : IN006A.S07                     **
;*  MODULE DEF. : UTILITIES OF INIT. MODULE      **
;*  DATE        : JULY 31 1986                   **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
        NAME    UTINIT
;
        PUBLIC  MOVBD,WRWAIT
        PUBLIC  FILZER
        PUBLIC  INZERO
;
CNT1    EQU     15
CNT2    EQU     145
;
        CSEG
;
;*************************************************
;  PROCEDURE : MOVBD
;  FUNCTION  : MOVES THE DATA STRING TO POINTED RAM
;  PLM CALL  :  ;CALL MOVBD(.SOURCE,LEN,.DEST);
;  INPUT     : .SOURCE  WORD  (ADDRESS OF DATA)
;              LEN      BYTE  (LENGTH OF DATA)
;              .DEST    WORD  (ADDRESS OF DESTIN.)
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : NONE
;  DESTROYS  : ALL
;
MOVBD:
        POP     H
        XTHL            ; take source adr. from stack
MDAT  : MOV     A,M
        XCHG
        MOV     M,A     ; put data into memory
        XCHG
        INX     H
        INX     D
        DCR     C
        JNZ     MDAT
        RET
;
;*************************************************
;  PROCEDURE : WRWAIT
;  FUNCTION  : DELAY OF 10 msec
;  PLM CALL  :   ; CALL WRWAIT;
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : PULSE
;  DESTROYS  : A
;
```

```
MRWAIT:
        PUSH    D
        MVI     D,CNT1 ; first loop's couter
LOOP1:  MVI     E,CNT2 ; second loop's counter
LOOP2:  DCR     E
        JNZ     LOOP2
        DCR     D
        JNZ     LOOP1
        POP     D
;
        RET
;
;*************************************************
;  PROCEDURE : FILZER
;  FUNCTION  : FILLs THE GIVEN BLOCK BY ZEROS
;  PLM CALL  :  CALL FILZER(START$ADDR,CNT)
;  INPUT     : [BC] START ADDRESS OF BLOCK
;              [DE]  LENGTH OF BLOCK
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : NONE
;  DESTROYS  : D,E,H,L,B,C
;
FILZER:
        MOV     L,C
        MOV     H,B
        MOV     A,D
        ORA     E
        RZ
;
GOON:   MVI     M,0
        INX     H
        DCX     D
        MOV     A,D
        ORA     E
        JNZ     GOON
;
        RET
;
;*************************************************
;  PROCEDURE : INZERO
;  FUNCTION  : FILLS THE RAM AREA WITH ZERO
;  PLM CALL  :  CALL INZERO
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : NONE
;  DESTROYS  : ?
;
INZERO:
        LXI     H,0F800H
ZERO:   MVI     M,0
        INX     H
        MOV     A,H
        CPI     0FFH
        JNZ     ZERO
        MOV     A,L
        CPI     7FH
        JNZ     ZERO
;
```

```
        RET
;
        END
;
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*        UNIVERSAL MEASUREMENT SYSTEM           **
;*           CONTROLLER WITH GPIB                **
;*                                               **
;*              MASTER CONTROL                   **
;*                                               **
;*  FILE NAME   : MC001A.S11                     **
;*  MODULE DEF. : MASTER CONTROL OF SYSTEM       **
;*  DATE        : AUGUST 07 1986                 **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
        NAME    MASCON
;
        PUBLIC  MASTER
        PUBLIC  MESOUT,TBLSER
        PUBLIC  GETCH,CRTOUT
        PUBLIC  EMPTY
        PUBLIC  EXRECV,EXSEND
;
        EXTRN   EDCMD,LSTCMD
        EXTRN   CPTCMD
;
        EXTRN   SEND,RECV
;
        EXTRN   MONXDT,CONIN,CONOUT
        EXTRN   MSEXIT,MONFIL
;
        EXTRN   DISBUF,LLIST,TLIST
        EXTRN   OBUF,INBUF,OBFCNT
;
        CSEG
;
;*************************************************
; PROCEDURE : MASTER
; FUNCTION  : CONTROLS THE BACKGROUND CYCLE
; PLM CALL  : NONE
; INPUT     : NONE
; OUTPUT    : NONE
; GLOBALS   : NONE
; CALLS     : MONXDT
; DESTROYS  : ALL
;
MASTER:
        LXI     H,READY
        CALL    MESOUT
LOOP:   LXI     H,PROMPT
        CALL    MESOUT
        CALL    GETCH
        LXI     H,PCTBL
        CALL    TBLSER
        JC      ERROR
        MOV     A,M
        INX     H
        MOV     H,M
        MOV     L,A
        LXI     B,RETADR
```

```
        PUSH    B
        PCHL
;
RETADR: JNC     LOOP
;
ERROR:  LXI     H,ERRMSG
        CALL    MESOUT
        JMP     LOOP
;
ERRMSG: DB      ODH,'SYSTEM ERROR !',0
;
READY:  DB      ODH,'GPIB SYSTEM IS READY',0
;
PROMPT: DB      ODH,'#',0
;
;***************************************************
;
PCTBL:  DB      4
        DW      3
        DB      'M'
        DW      MONCMD
        DB      'E'
        DW      EDCMD
        DB      'L'
        DW      LSTCMD
        DB      'C'
        DW      CPTCMD
;
;***************************************************
;  PROCEDURE : MESOUT
;  FUNCTION  : TYPE A MESSAGE FROM CRT
;  PLM CALL  : NONE
;  INPUT     :
;  OUTPUT    :
;  GLOBALS   :
;  CALLS     :
;  DESTROYS  :
;
MESOUT:
        MOV     A,M
        ORA     A
        RZ
;
        MOV     C,A
        CALL    CONOUT
        INX     H
        JMP     MESOUT
;
;***************************************************
;  PROCEDURE : GETCH
;  FUNCTION  : GETS A CHARACTER FROM MONITOR
;  PLM CALL  :
;  INPUT     :
;  OUTPUT    :
;  GLOBALS   :
;  CALLS     :
;  DESTROYS  :
;
GETCH:
        CALL    CONIN
```

```
        JZ      GETCH
        ANI     7FH
;
        RET
;
;*****************************************************
;  PROCEDURE : TBLSER
;  FUNCTION  : SEARH COMMAND ADDRESS IF IT IS VALID
;  PLM CALL  :
;  INPUT     :
;  OUTPUT    :
;  GLOBALS   :
;  CALLS     :
;  DESTROYS  :
;
TBLSER:
        MOV     B,M
        INX     H
        MOV     E,M
        INX     H
        MOV     D,M
        INX     H
;
PTSER:  CMP     M
        JZ      PTFIN
        DAD     D
        DCR     B
        JNZ     PTSER
        STC
;
PTFIN:  INX     H
;
        RET
;*****************************************************
;  PROCEDURE : MONCMD
;  FUNCTION  : CALLS MONITOR
;  PLM CALL  :
;  INPUT     :
;  OUTPUT    :
;  GLOBALS   :
;  CALLS     :
;  DESTROYS  :
;
MONCMD:
        XRA     A
        STA     MSEXIT
        STA     MONFIL
;
INMON:  CALL    MONXOT
        LDA     MSEXIT
        CPI     1
        JNZ     INMON
;
        RET
;
;*****************************************************
;  PROCEDURE : CRTOUT
;  FUNCTION  : TYPE MESSAGE TO CRT
;  PLM CALL  : CALL MESSAGE(.MESSAGE)
;  INPUT     : [BC] ADDRESS OF MESSAGE
```

```
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : MESOUT
;  DESTROYS  : HL,BC,
;
CRTOUT:
        MOV     L,C
        MOV     H,B
;
        CALL    MESOUT
;
        RET
;
;*****************************************
;  PROCEDURE: EMPTY
;  FUNCTION : CHECKS THE GIVEN LINE
;  PLM CALL : EMPTY(ADD)
;  INPUTS   : [BC]... START ADDRESS
;  OUTPUTS  : [A]..0 EMPTY LINE
;  GLOBALS  : NONE
;  CALLS    : NONE
;  DESTROYS : A,HL,BC,E
;
EMPTY:
        MOV     L,C
        MOV     H,B
        MVI     E,32
        XRA     A
        MOV     A,M
;
NOTFIN: INX     H
        DCR     E
        RZ
;
        ORA     M
        JMP     NOTFIN
;
;*************************************************
;  PROCEDURE : EXSEND
;  FUNCTION  :
;  PLM CALL  :
;  INPUT     :
;  OUTPUT    :
;  GLOBALS   :
;  CALLS     :
;  DESTROYS  :
;
EXSEND:
        LXI     H,LLIST
        LXI     D,OBUF
        LDA     OBFCNT
        MOV     C,A
        MVI     B,0AH
;
        CALL    SEND
;
        RET
;
;*************************************************
;  PROCEDURE : EXRECV
```

```
;  FUNCTION  :
;  PLM CALL  :
;  INPUT     :
;  OUTPUT    :
;  GLOBALS   :
;  CALLS     :
;  DESTROYS  :
;
EXRECV:
        LXI     H,TLIST
        LXI     D,INBUF
        MVI     C,32
        MVI     B,0AH
;
        CALL    RECV
;
        RET
;
        END
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*         UNIVERSAL MEASUREMENT SYSTEM          **
;*             CONTROLLER WITH GPIB              **
;*                                               **
;*                PRINTER SENDER                 **
;*                                               **
;*  FILE NAME   : PRTSEN.S02                     **
;*  MODULE DEF. : PRINTER SENDER                 **
;*  DATE        : JULY 31 1986                   **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
        NAME    SENDER
;
        PUBLIC  PRSEND
;
        EXTRN   USRTPC,USRTPD
;
        EXTRN   LINPTR,COLPTR,DSRCNT,FAILCT
;
        EXTRN   CARFLG,OBUFER,STAFL
;
        CSEG


;*************************************************
;  PROCEDURE : PRESEND
;  FUNCTION  : Sends the prepared lines in OBUFER
;              to the printer,one character each
;              time.
;  PLM CALL  : NONE
;  INPUT     : NONE
;  OUTPUT    : NONE
;  CALLS     : NONE
;  DESTROYS  : ALL
;*
;
PRSEND: IN LOW USRTPC   ;check usart3 status
        MOV     C,A     ;save for later use
        ANI     01H     ;IF usart is not ready
        RZ              ;...then RETURN
;
        MOV     A,C     ;get status byte again
        ANI     80H     ;if usart is ready (DSR bit=1)
        JNZ     CHKOFF  ;...then proceed to CHKOFF
        MVI     A,5     ;...else set DSRCNT off
        STA     DSRCNT  ;...to check later
        LDA     FAILCT  ;cneck fail counter
        ORA     A       ;prepare Z flag
        RZ              ;if alraedy zero then return
;
        DCR     A       ;else decrement fail counter
        STA     FAILCT  ;...to give alarm
        RET             ;then return
;
CHKOFF: LDA     DSRCNT  ;check previous state of DSR
        ORA     A       ;prepare Z flag
```

```
        JZ      GOSEND  ;if zero ,proceed to send bytes
        DCR     A       ;else decrement DSR counter
        STA     DSRCNT  ;save it to DSRCNT and
        RNZ             ;...if not zero,return
        MVI     A,0AH   ;send a LF for
        OUT LOW USRTPD  ;...printer recovery
        RET
;
GOSEND: LDA     FAILCT  ;check if alarm is given before
        ORA     A       ;prepare Z flag
        JNZ     CAL1    ;if not zero proceed to call
        MVI     A,0FFH  ;and set fail counter
        STA     FAILCT  ;...to FF
CAL1:   LDA     LINPTR  ;get line pointer
        MOV     B,A     ;save to B for later use
        CPI     0FFH    ;check if FF
        JZ      SCAN2   ;if so go to scan all status bytes
        LDA     COLPTR  ;check if line ended
        CPI     80      ;if not go on sending
        JNZ     BASLAT
        LDA     CARFLG  ;get 'carriage return' flag
        ORA     A       ;if CR already sent
        JNZ     BITIR
        MVI     A,0DH   ;else output 0D (carriage return)
        OUT LOW USRTPD  ;...from USART3
        MVI     A,1     ;set 'carriage return' flag
        STA     CARFLG  ;...to 1 next cycle
        RET
;
BASLAT: LXI     H,OBUFER;HL has address of buffer
        LXI     D,80    ;DE has 80
        MOV     A,B     ;A has line pointer
LOOP1:  ORA     A       ;prepare Z flag
        JZ      CAL2    ;if line pointer is zero proceed
        DAD     D       ;if not add 80
        DCR     A       ;...and decrement line pointer
        JMP     LOOP1   ;go to LOOP1 to check again
CAL2:   LDA     COLPTR  ;now HL shows the required buffer
        MOV     E,A     ;...get column pointer,save to E
        MVI     D,0     ;clear D
        DAD     D       ;now HL=HL+column pointer
        MOV     A,M     ;get that character
        OUT LOW USRTPD  ;send to USART3
        INR     E       ;increment column pointer
        MOV     A,E     ;and save it again
        STA     COLPTR  ;...to COLPTR
        RET
;
BITIR:  MVI     A,0AH   ;output 0A (line feed)
        OUT LOW USRTPD  ;...from USART3
        LXI     H,STAFL ;HL has status file begining
        MOV     E,B     ;get line pointer (B) to E
        MVI     D,0     ;clear D
        DAD     D       ;HL has status byte of line pointed
        XRA     A       ;clear A to 0 to
        MOV     M,A     ;...store to that status byte
        STA     COLPTR  ;also clear column pointer
        STA     CARFLG  ;...and CR flag
        MVI     A,0FFH  ;set linepointer to FF
        STA     LINPTR  ;to mean no line being printed
```

```
        RET
;
SCAN2:  LXI     H,STAFL ;HL now shows begining of STAFL
        MVI     D,0     ;D is now index register ,cleared
LOOP2:  MOV     A,M     ;get that status byte
        CPI     2       ;if that byte is 2
        JZ      FOUND   ;...then proceed to FOUND
        INX     H       ;else access next status byte
        INR     D       ;increment counter
        MOV     A,D     ;check if the counter has
        CPI     6       ;...reached 6
        JNZ     LOOP2   ;if not yet,go to LOOP1
        RET
;
FOUND:  MOV     A,D     ;D has line number ,now in A also
        STA     LINPTR  ;line number in LINPTR
        MVI     A,3     ;store 3 to the status byte
        MOV     M,A     ;...of that line
        XRA     A       ;clear A to 0 to
        STA     COLPTR  ;...clear column pointer and
        RET
;
;****************************************
;
        END
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*        UNIVERSAL MEASUREMENT SYSTEM           **
;*            CONTROLLER WITH GPIB               **
;*                                               **
;*          FOREGROUND SCHEDULER                 **
;*                                               **
;*  FILE NAME   : FG001A.S04                     **
;*  MODULE DEF. : SERVICE OF RST 7.5             **
;*  DATE        : APRIL 13 1986                  **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
        NAME    FGSERV
;
        EXTRN   PRTCTR
;
        EXTRN   PRSEND
;
        PUBLIC  RST75
;
        CSEG
;
;*************************************************
;  PROCEDURE : RST75
;  FUNCTION  : ACTIVITES FOREGROUND TASKS
;  PLM CALL  : NONE
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : SYSFIL
;  CALLS     : NONE
;  DESTROYS  : NONE
;
RST75:
        PUSH    PSW
        PUSH    B
        PUSH    D
        PUSH    H
;
        LDA     PRTCTR
        INR     A
        CPI     8
        JNZ     EXIT
;
        CALL    PRSEND
;
        XRA     A
;
EXIT :  STA     PRTCTR
;
        MVI     A,10H
        SIM
;
        POP     H
        POP     D
        POP     B
        POP     PSW
;
```

```
        EI
;
        RET
;
;***************************************
;
        END
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*        UNIVERSAL MEASUREMENT SYSTEM           **
;*          CONTROLLER WITH GPIB                 **
;*                                               **
;*                  EDITOR                       **
;*                                               **
;*  FILE NAME   : ED001A.S03                     **
;*  MODULE DEF. : EXECUTIVE                      **
;*  DATE        : JUNE 15 1986                   **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
        NAME    EDITEXEC
;
        PUBLIC  EDCMD
;
        EXTRN   EDSCAN
;
        EXTRN   EDSTA
        EXTRN   IDLCTR
;
        LINESTR EQU     8000H
        NLINE   EQU     60
        NCHAR   EQU     32
;
        CSEG
;
;*************************************************
;  PROCEDURE : EDCMD
;  FUNCTION  : EXECUTE THE EDITOR ROUTINES
;  PLM CALL  : NONE
;  INPUT     : NONE
;  OUTPUT    : NONE
;  GLOBALS   : NONE
;  CALLS     : EDSCAN
;  DESTROYS  : ALL
;
EDCMD:
        MVI     A,1
        STA     EDSTA
;
        LXI     H,LINESTR
        MVI     E,NLINE
        MVI     C,NCHAR
        MVI     B,0
;
LOOP:   MOV     A,M
        CPI     0
        JZ      IDLFND
;
        DAD     B
        DCR     E
        JNZ     LOOP
;
NOIDL:  MVI     A,11
        STA     EDSTA
        JMP     SCAN
```

```
;
IDLFND: MOV     A,E
        STA     IDLCTR
;
;
SCAN:   CALL    EDSCAN
        LDA     EDSTA
        CPI     0
        RZ
;
        JMP     SCAN
;
        END
```

```
$PAGEWIDTH(80)
/*****************************************************
**                                                  **
**          UNIVERSAL MEASUREMENT SYSTEM            **
**              CONTROLLER WITH GPIB                **
**                                                  **
**                      EDITOR                      **
**                                                  **
**  FILE NAME   : ED002P.S12                        **
**  MODULE DEF. : REALIZE EDITOR FUNCTIONS          **
**  DATE        : JULY 13 1986                      **
**  AUTHOR      : ILHAN NEDIM ALP                   **
**                                                  **
*****************************************************/

EDITOR1:
DO;
DECLARE (EDSTA,
        COLUMN,
        LINE$NO,
        KEY,
        IDLCTR,
        REDSTA) BYTE EXTERNAL;

DECLARE LINE(1) STRUCTURE(
        LCHAR(32)       BYTE) EXTERNAL;

DECLARE WBUFER(32) BYTE EXTERNAL;

DECLARE CONV(6) BYTE EXTERNAL;

DECLARE DISBUF(33) BYTE EXTERNAL;

DECLARE (I,J,K) BYTE;

DECLARE NLINE           LITERALLY'60',
        NCHAR           LITERALLY'32',
        RUBOUTALL       LITERALLY'7FH',
        SPACE           LITERALLY'20H',
        ESCAPE          LITERALLY'1BH',
        CARRIAGE$RETURN LITERALLY'0DH',
        BEGIN$STATE     LITERALLY'PROCEDURE PUBLIC;',
        GET$KEY         LITERALLY'GETCH';

DECLARE ERROR(*) BYTE DATA (0DH,'ERROR',0);
DECLARE TRY$AGAIN(*) BYTE DATA (0DH,'PLEASE TRY AGAIN',0);
DECLARE PROMPT(*) BYTE DATA (0DH,'$',0);
DECLARE LINEFREE(*) BYTE DATA (' LINE(S) FREE',0);
DECLARE BEEP(*) BYTE DATA (07H,0);
DECLARE NOIDLELINE(*) BYTE DATA (0DH,'NO IDLE LINE',0);
DECLARE CR(*) BYTE DATA (0DH,0);
DECLARE SP(*) BYTE DATA (20H,0);
DECLARE LEDIT(*) BYTE DATA (0DH,'LINE EDITOR',0);
DECLARE MES1(*) BYTE DATA (0DH,'TYPE LINE NUMBER',0);
DECLARE MES2(*) BYTE DATA (0DH,'PRESS RUBOUT KEY FOR CLEAR ALL LINES',0);
DECLARE MES3(*) BYTE DATA (0DH,'PRESS SPACE FOR RETURN LINE EDITOR',0);
DECLARE MES4(*) BYTE DATA (0DH,'PRESS ANY KEY TO EXIT',0);
DECLARE DELETE(*) BYTE DATA (08H,20H,08H,0);
DECLARE EXCEEDED(*) BYTE DATA (0DH,'OUT OF RANGE',0);
DECLARE FULL(*) BYTE DATA (0DH,'NO MORE IDLE LINE',0);
```

```
DECLARE BS(*) BYTE DATA (08H,0);
DECLARE WAIT(*) BYTE DATA (0DH,'WAIT',0);

BINASC: PROCEDURE(NUMBER) EXTERNAL;
  DECLARE NUMBER BYTE;
  END BINASC;

CRTOUT: PROCEDURE(MSG$ADR) EXTERNAL;
  DECLARE MSG$ADR ADDRESS;
  END CRTOUT;

WRWAIT:PROCEDURE EXTERNAL;
  END WRWAIT;

CHECK$KEY: PROCEDURE(KEYNO) BYTE EXTERNAL;
  DECLARE KEYNO BYTE;
  END CHECK$KEY;

GETCH: PROCEDURE BYTE EXTERNAL;
  END GETKEY;

FILZER: PROCEDURE(STR$PTR,CNT) EXTERNAL;
  DECLARE STR$PTR ADDRESS;
  DECLARE CNT ADDRESS;
  END FILZER;

MOVBD: PROCEDURE(SRC$ADR,LENGTH,DES$ADR) EXTERNAL;
  DECLARE (SRC$ADR,DES$ADR) ADDRESS;
  DECLARE LENGTH BYTE;
  END MOVBD;

/*****************************************
*
*  PROCEDURE : PUT$CHR
*  FUNCTION  : PUTS THE CHAR. INTO MBUFER
*  PLM CALL  : CALL PUT$CHR
*  INPUTS    : NONE
*  OUTPUTS   : NONE
*  GLOBALS   : MBUFER,EDFIL
*  CALLS     : CRTOUT
*/

PUT$CHR: PROCEDURE PUBLIC;

  MBUFER(COLUMN) = KEY;
  COLUMN = COLUMN+1;
  IF COLUMN >= 30 THEN
    CALL CRTOUT(.BEEP);
  RETURN;

END PUT$CHR;

/*************************************************
*  PROCEDURE : TRY
*  FUNCTION  : SEND ERROR MESSAGES TO CRT
*  PLM CALL  : CALL TRY
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : NONE
*  CALLS     : CRTOUT
```

```
*/

TRY: PROCEDURE PUBLIC;

  CALL CRTOUT(.ERROR);
  CALL CRTOUT(.TRY$AGAIN);
  CALL CRTOUT(.PROMPT);
  EDSTA=4;
  RETURN;

END TRY;

/*********************************************
*  STATE    : EDS001
*  FUNCTION :
*  ENTERS   : EDSCAN
*  EXITS    : EDS002,EDS004,EDS005
*
*/
EDS001: BEGIN$STATE

  CALL CRTOUT(.LEDIT);
  CALL CRTOUT(.CR);
  CALL FILZER(.CONV(0),6);
  CALL BIN$SC(IDLCTR);
  CALL CRTOUT(.CONV(0));
  CALL CRTOUT(.LINEFREE);
  CALL CRTOUT(.MES1);
  CALL CRTOUT(.PROMPT);
  CALL CRTOUT(.SP);
  EDSTA=4;
  KEY=0;
  RETURN;
END EDS001;

/*********************************************
*  STATE    : EDS004
*  FUNCTION : GETS THE FIRST DIGIT OF LINE NUMBER
*  ENTERS   : EDS001,EDS005
*  EXITS    : EDS005
*
*/
EDS004: BEGIN$STATE

  KEY=GETKEY;
  K=CHECK$KEY(KEY);
  LINE$NO=0;
  DO CASE K;
    /* Other (0) */
  CALL TRY;
    /* Back Space (1) */
  CALL CRTOUT(.SP);
    /* Carriage Return (2) */
  DO;
    LINE$NO=0;
    EDSTA=6;
  END;
    /* Escape (3) */
  EDSTA=0;
    /* Space Bar (4) */
```

```
  CALL CRTOUT(.BS);
    /* Characters (5) */
  CALL CRTOUT(.DELETE);
    /* Digits (6) */
  DO;
    LINE$NO=(KEY AND 0FH)*10;
    EDSTA=5;
  END;
    /* Rubout (7) */
  CALL TRY;
    /* Cursor Right (8) */
  CALL CRTOUT(.BS);
  END;   /* END OF DO CASE */
  KEY=0;
  RETURN;
END EDS004;

/*********************************************
*  STATE    : EDS005
*  FUNCTION : GET SECOND DIGIT OF LINE NUMBER
*  ENTERS   : EDS004
*  EXITS    : EDS004,EDS006
*
*/
EDS005: BEGIN$STATE
  KEY=GETKEY;
  K=CHECK$KEY(KEY);
  DO CASE K;
    /* Other (0) */
  CALL TRY;
    /* Back Space (1) */
  DO;
    CALL CRTOUT(.SP);
    CALL CRTOUT(.DELETE);
    EDSTA=4;
  END;
    /* Carriage Return (2) */
  DO;
    LINENO=LINE$NO/10;
    EDSTA=6;
  END;
    /* Escape (3) */
  EDSTA=0;
    /* Space Bar (4) */
  CALL CRTOUT(.BS);
    /* Characters (5) */
  CALL CRTOUT(.DELETE);
    /* Digits (6) */
  DO;
    LINE$NO=LINE$NO + (KEY AND 0FH);
    IF LINE$NO > 59 THEN
      DO;
        CALL CRTOUT(.ERROR);
        CALL CRTOUT(.EXCEEDED);
        CALL CRTOUT(.TRY$AGAIN);
        CALL CRTOUT(.PROMPT);
        EDSTA=4;
      END;
    ELSE
      EDSTA=6;
```

```
  END;
    /* Rubout (7) */
  CALL TRY;
    /* Cursor Right (8) */
  CALL CRTOUT(.BS);
  END;  /* END OF DO CASE */
  KEY=0;
  RETURN;
END EDS005;

/*********************************************
*  STATE    : EDS006
*  FUNCTION : PREPARATION STATE FOR LINE EDITING
*  ENTERS   : EDS005
*  EXITS    : EDS007
*
*/
EDS006: BEGIN$STATE
  CALL CRTOUT(.CR);
  CALL FILZER(.CONV(0),6);
  CALL BINASC(LINE$NO);
  CALL CRTOUT(.SP);
  CALL CRTOUT(.CONV(0));
  CALL CRTOUT(.SP);
  CALL FILZER(.DISBUF(0),33);
  CALL MOVBD(.LINE(LINE$NO),32,.DISBUF(0));
  CALL CRTOUT(.DISBUF(0));
  CALL CRTOUT(.PROMPT);
  CALL CRTOUT(.CONV(0));
  CALL CRTOUT(.SP);
  CALL FILZER(.WBUFER(0),NCHAR);
  COLUMN=0;
  REDSTA=6;
  EDSTA=7;
  RETURN;
END EDS006;

/*********************************************
*  STATE    : EDS007
*  FUNCTION : COLLECT CHARACTERS
*  ENTERS   : EDS006,EDS002
*  EXITS    : EDS008
*
*/
EDS007: BEGIN$STATE

  IF COLUMN > 31 THEN
    DO;
      EDSTA=8;
      RETURN;
    END;
  KEY=GET$KEY;
  K=CHECK$KEY(KEY);
  DO CASE K;
    /* Others (0) */
  RETURN;
    /* Back Space (1) */
  DO;
    IF COLUMN <> 0 THEN
      DO;
```

```
        COLUMN = COLUMN - 1;
        WBUFER(COLUMN) = 0;
      END;
    ELSE
      CALL CRTOUT(.SP);
  END;
    /* Carriage Return (2) */
  DO;
    IF COLUMN=0 THEN
      CALL MOVBD(.LINE(LINE$NO).LCHAR(0),32,.WBUFER(0));
    ELSE
      DO I=COLUMN TO 31;
        WBUFER(I)=0;
      END;
    EDSTA=8;
  END;
    /* Escape (3) */
  DO;
    IF COLUMN <> 0 THEN
      DO;
        DO I=COLUMN TO 31;
          WBUFER(I) = LINE(LINE$NO).LCHAR(I);
        END;
        REDSTA=0;
        EDSTA=8;
      END;
    ELSE
      EDSTA=0;
  END;
    /* Space Bar (4) */
  CALL PUT$CHR;
    /* Characters (5) */
  CALL PUT$CHR;
    /* Digits (6) */
  CALL PUT$CHR;
    /* Rubout (7) */
  DO;
    DO I=0 TO 31;
      WBUFER(I) = 0;
    END;
    EDSTA=8;
  END;
    /* Cursor Right (8) */
  DO;
    CALL CRT$OUT(.BS);
    IF LINE(LINE$NO).LCHAR(COLUMN) <> 0 THEN
      DO;
        KEY=LINE(LINE$NO).LCHAR(COLUMN);
        DISBUF(0)=KEY;
        DISBUF(1)=0;
        CALL CRTOUT(.DISBUF(0));
        CALL PUT$CHR;
      END;
  END;
  END;   /* END OF DO CASE */
  KEY=0;
  RETURN;

END EDS007;
```

```
/*********************************************
* STATE    : EDS008
* FUNCTION : WRITE INTO THE LINE
* ENTERS   : EDS007
* EXITS    : EDS009
*
*/
EDS008: BEGIN$STATE
  DO I=0 TO 31;
   LINE(LINE$NO).LCHAR(I)=WBUFER(I);
   CALL WRWAIT;
  END;
  IF REDSTA = 0 THEN
   DO;
     LINE$NO =0;
     EDSTA=0;
     RETURN;
   END;
  LINE$NO=LINE$NO+1;
  IF LINE$NO ) 59 THEN
    EDSTA=9;
  ELSE
    EDSTA=6;
  RETURN;
END EDS008;
/*********************************************
* STATE    : EDS009
* FUNCTION :
* ENTERS   :
* EXITS    :
*
*/
EDS009: BEGIN$STATE
  CALL CRTOUT(.FULL);
  CALL CRTOUT(.MES3);
  CALL CRTOUT(.MES4);
  CALL CRTOUT(.PROMPT);
  KEY=GETKEY;
  IF KEY = SPACE THEN
   DO;
     CALL CRTOUT(.MES1);
     CALL CRTOUT(.PROMPT);
     EDSTA=4;
   END;
  ELSE
    EDSTA=0;
  RETURN;
END EDS009;

/*********************************************
* STATE    : EDS010
* FUNCTION :
* ENTERS   :
* EXITS    :
*
*/
EDS010: BEGIN$STATE
  DO I=0 TO NLINE-1;
   DO J=0 TO NCHAR-1;
     LINE(I).LCHAR(J)=0;
```

```
      CALL WRWAIT;
    END;
  END;
  CALL CRTOUT(.MES3);
  CALL CRTOUT(.MES4);
  CALL CRTOUT(.PROMPT);
  KEY=GET$KEY;
  IF KEY = SPACE THEN
    EDSTA=1;
  ELSE
    EDSTA=0;
  IDLCTR=60;
  KEY=0;
  RETURN;
END EDS010;

/*********************************************
*  STATE    : EDS011
*  FUNCTION :
*  ENTERS   :
*  EXITS    :
*
*/
EDS011: BEGIN$STATE
  CALL CRTOUT(.MD$IDLE$LINE);
  CALL CRTOUT(.MES2);
  CALL CRTOUT(.MES3);
  CALL CRTOUT(.MES4);
  CALL CRTOUT(.PROMPT);
  EDSTA=0;
  KEY=GETKEY;
  IF KEY=SPACE THEN
    DO;
      IDLCTR=0;
      EDSTA=1;
    END;
  IF KEY=RUBOUTALL THEN
    DO;
      CALL CRTOUT(.WAIT);
      EDSTA=10;
    END;
  KEY=0;
  RETURN;
END EDS011;

/*****************************************************
*  PROCEDURE : EDSCAN
*  FUNCTION  : ROUTES THE EDITING STATES
*  PLM CALL  : NONE
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : EDSCAN
*  CALLS     : ALL EDS...
*/
EDSCAN: PROCEDURE PUBLIC;

  DO CASE EDSTA;
    RETURN;
    CALL EDS001;
    RETURN;
```

```
    RETURN;
    CALL EDS004;
    CALL EDS005;
    CALL EDS006;
    CALL EDS007;
    CALL EDS008;
    CALL EDS009;
    CALL EDS010;
    CALL EDS011;
  END;
END EDSCAN;

/*****************************************/

END EDITOR1;
```

```
$PAGEWIDTH(80)
/*****************************************************
**                                                  **
**        UNIVERSAL MEASUREMENT SYSTEM              **
**            CONTROLLER WITH GPIB                  **
**                                                  **
**                    LIST                          **
**                                                  **
** FILE NAME   : LS001P.S04                         **
** MODULE DEF. : REALIZE LIST FUNCTIONS             **
** DATE        : JULY 09 1986                       **
** AUTHOR      : ILHAN NEDIM ALP                    **
**                                                  **
*****************************************************/

LIST1:
DO;
DECLARE (EDSTA,
        COLUMN,
        LINE$NO,
        KEY,
        IDLCTR,
        REDSTA,
        RI,
        FIRST,
        LAST) BYTE EXTERNAL;

DECLARE LINE(1) STRUCTURE(
        LCHAR(32)       BYTE) EXTERNAL;

DECLARE CONV(6) BYTE EXTERNAL;

DECLARE DISBUF(33) BYTE EXTERNAL;

DECLARE (I,K) BYTE;

DECLARE RUBOUTALL       LITERALLY'7FH',
        SPACE           LITERALLY'20H',
        ESCAPE          LITERALLY'1BH',
        CARRIAGE$RETURN LITERALLY'0DH',
        BEGIN$STATE     LITERALLY'PROCEDURE PUBLIC;',
        GET$KEY         LITERALLY'GETCH';

DECLARE ERROR(*) BYTE DATA (0DH,'ERROR',0);
DECLARE CR(*) BYTE DATA (0DH,0);
DECLARE SP(*) BYTE DATA (20H,0);
DECLARE EXCEEDED(*) BYTE DATA (0DH,'OUT OF RANGE',0);
DECLARE BS(*) BYTE DATA (08H,0);
DECLARE DELETE(*) BYTE DATA (08H,20H,08H,0);
DECLARE LIST(*) BYTE DATA (0DH,'LIST FROM ',0);
DECLARE T(*) BYTE DATA (' TO ',0);

BINASC: PROCEDURE(NUMBER) EXTERNAL;
  DECLARE NUMBER BYTE;
  END BINASC;

CRTOUT: PROCEDURE(MSG$ADR) EXTERNAL;
  DECLARE MSG$ADR ADDRESS;
  END CRTOUT;
```

```
CHECK$KEY: PROCEDURE(KEYNO) BYTE EXTERNAL;
  DECLARE KEYNO BYTE;
  END CHECK$KEY;

GETCH: PROCEDURE BYTE EXTERNAL;
  END GETKEY;

FILZER: PROCEDURE(STR$PTR,CNT) EXTERNAL;
  DECLARE STR$PTR ADDRESS;
  DECLARE CNT ADDRESS;
  END FILZER;

MOVBD: PROCEDURE(SRC$ADR,LENGTH,DES$ADR) EXTERNAL;
  DECLARE (SRC$ADR,DES$ADR) ADDRESS;
  DECLARE LENGTH BYTE;
  END MOVBD;

/*********************************************
*  STATE    : LSS001
*  FUNCTION :
*  ENTERS   : LSSCAN
*  EXITS    : LSS002
*
*/
LSS001: BEGIN$STATE

  CALL CRTOUT(.LIST);
  FIRST=0;
  LAST=0;
  RI=0;
  KEY=0;
  REDSTA=4;
  EDSTA=2;
  RETURN;
END LSS001;

/*********************************************
*  STATE    : LSS002
*  FUNCTION : GETS THE FIRST DIGIT OF LINE NUMBER
*  ENTERS   : LSS001,LSS004
*  EXITS    : LSS003,LSS001
*
*/
LSS002: BEGIN$STATE

  KEY=GETKEY;
  K=CHECK$KEY(KEY);
  LINE$NO=0;
  DO CASE K;
    /* Other (0) */
  DO;
    CALL CRTOUT(.ERROR);
    EDSTA=1;
  END;
    /* Back Space (1) */
  CALL CRTOUT(.SP);
    /* Carriage Return (2) */
  DO;
    IF REDSTA=4 THEN
      FIRST=0;
```

```
    LAST=59;
    EDSTA=5;
  END;
    /* Escape (3) */
  EDSTA=0;
    /* Space Bar (4) */
  CALL CRTOUT(.BS);
    /* Characters (5) */
  CALL CRTOUT(.DELETE);
    /* Digits (6) */
  DO;
    LINE$NO=(KEY AND 0FH)*10;
    EDSTA=3;
  END;
    /* Rubout (7) */
  DO;
    CALL CRTOUT(.ERROR);
    EDSTA=1;
  END;
    /* Cursor Right (8) */
  DO;
    CALL CRTOUT(.ERROR);
    EDSTA=1;
  END;
  END;   /* END OF DO CASE */
  KEY=0;
  RETURN;
END LSS002;

/*********************************************
*  STATE    : LSS003
*  FUNCTION : GET SECOND DIGIT OF LINE NUMBER
*  ENTERS   : LSS002
*  EXITS    : LSS001,LSS004
*
*/
LSS003: BEGIN$STATE
  KEY=GETKEY;
  K=CHECK$KEY(KEY);
  DO CASE K;
    /* Other (0) */
  DO;
    CALL CRTOUT(.ERROR);
    EDSTA=1;
  END;
    /* Back Space (1) */
  DO;
    CALL CRTOUT(.SP);
    CALL CRTOUT(.DELETE);
    EDSTA=2;
  END;
    /* Carriage Return (2) */
  DO;
    LINENO=LINENO/10;
    IF REDSTA=4 THEN
      FIRST=LINENO;
    ELSE
      LAST=LINENO;
    EDSTA=5;
  END;
```

```
    /* Escape (3) */
  EDSTA=0;
    /* Space Bar (4) */
  CALL CRTOUT(.BS);
    /* Characters (5) */
  CALL CRTOUT(.DELETE);
    /* Digits (6) */
  DO;
    LINE$NO=LINE$NO + (KEY AND 0FH);
    IF REDSTA=4 THEN
      DO;
        FIRST=LINENO;
        EDSTA=4;
      END;
    ELSE
      DO;
        LAST=LINENO;
        EDSTA=5;
      END;
  END;
    /* Rubout (7) */
  DO;
    CALL CRTOUT(.ERROR);
    EDSTA=1;
  END;
    /* Cursor Right (8) */
  DO;
    CALL CRTOUT(.ERROR);
    EDSTA=1;
  END;
  END;   /* END OF DO CASE */
  KEY=0;
  RETURN;
END LSS003;

/*********************************************
*  STATE    : LSS004
*  FUNCTION : PREP.STATE FOR GETTING DEST
*  ENTERS   : LSS003
*  EXITS    : LSS002
*
*/
LSS004: BEGIN$STATE
  CALL CRTOUT(.T);
  REDSTA=5;
  EDSTA=2;
  RETURN;
END LSS004;

/*********************************************
*  STATE    : LSS005
*  FUNCTION : PREP. STATE FOR DISPLAY LINES
*  ENTERS   : LSS002,LSS003
*  EXITS    : LSS006
*
*/
LSS005: BEGIN$STATE

  IF LAST ) FIRST AND
     FIRST (= 59 THEN
```

```
    DO;
      IF LAST > 59 THEN
        LAST=59;
      RI=FIRST;
      EDSTA=6;
    END;
  ELSE
    DO;
      CALL CRTOUT(.ERROR);
      EDSTA=1;
    END;
  RETURN;

END LSS005;

/*****************************************
*  STATE    : LSS006
*  FUNCTION : DISPLAY STATE
*  ENTERS   : LSS005
*  EXITS    :
*
*/
LSS006: BEGIN$STATE

DO I=RI TO LAST;
   CALL FILZER(.DISBUF,33);
   CALL MOVBD(.LINE(I),32,.DISBUF(0));
   CALL BINASC(I);
   CALL CRTOUT(.CR);
   CALL CRTOUT(.SP);
   CALL CRTOUT(.CONV(0));
   CALL CRTOUT(.SP);
   CALL CRTOUT(.DISBUF(0));

  KEY=GETKEY;
  K=CHECK$KEY(KEY);
  DO CASE K;
    /* Other (0) */
  ;
    /* Back Space (1) */
  CALL CRTOUT(.SP);
    /* Carriage Return (2) */
  ;
    /* Escape (3) */
  DO;
    EDSTA=0;
    RETURN;
  END;
    /* Space Bar (4) */
  CALL CRTOUT(.BS);
    /* Characters (5) */
  DO;
    CALL CRTOUT(.DELETE);
    IF KEY='A' THEN
      DO;
        EDSTA=5;
        RETURN;
      END;
    IF KEY='T' THEN
      DO;
```

```
        FIRST=0;
        LAST=59;
        EDSTA=5;
        RETURN;
      END;
  END;
    /* Digits (6) */
  CALL CRTOUT(.DELETE);
    /* Rubout (7) */
  ;
    /* Cursor Right (8) */
  ;
  END;   /* END OF DO CASE */
  KEY=0;
END;
EDSTA=0;
RETURN;

END LSS006;

/*****************************************************
*  PROCEDURE : LSSCAN
*  FUNCTION  : ROUTES THE LISTING STATES
*  PLM CALL  : NONE
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : LSSCAN
*  CALLS     : ALL LSS...
*/
LSSCAN: PROCEDURE PUBLIC;

  DO CASE EDSTA;
    RETURN;
    CALL LSS001;
    CALL LSS002;
    CALL LSS003;
    CALL LSS004;
    CALL LSS005;
    CALL LSS006;
  END;
END LSSCAN;

/*****************************************************
*  PROCEDURE : LSTCMD
*  FUNCTION  : EXECUTIVE OF LISTING
*  PLM CALL  :
*  INPUT     :
*  OUTPUT    :
*  GLOBALS   :
*  CALLS     :
*/

LSTCMD: PROCEDURE PUBLIC;

  EDSTA=1;
LSLOOP:
  CALL LSSCAN;
  IF EDSTA <> 0 THEN
    GOTO LSLOOP;
  RETURN;
```

```
END LSTCMD;

/*****************************************/

END LIST1;
```

```
$PAGEWIDTH(132)
/*****************************************************
**                                                  **
**          UNIVERSAL MEASUREMENT SYSTEM            **
**              CONTROLLER WITH GPIB                **
**                                                  **
**                    COMPILER                      **
**                                                  **
**  FILE NAME   : CP001P.S07                        **
**  MODULE DEF. : COMPILATION ROUTINES              **
**  DATE        : JULY 28 1986                      **
**  AUTHOR      : ILHAN NEDIM ALP                   **
**                                                  **
*****************************************************/

COMP1:
DO;
DECLARE (EDSTA,
        COLUMN,
        LINE$NO,
        KEY,
        IDLCTR,
        REDSTA,
        RI,
        FIRST,
        LAST,
        ENDFLG,
        JMPFLG) BYTE EXTERNAL;

DECLARE LINE(1) STRUCTURE(
        LCHAR(32)       BYTE) EXTERNAL;

DECLARE CONV(6) BYTE EXTERNAL;

DECLARE BEGIN$STATE     LITERALLY'PROCEDURE PUBLIC;',
        GET$KEY         LITERALLY'GETCH';

DECLARE CR(*) BYTE DATA (0DH,0);
DECLARE COMP(*) BYTE DATA (0DH,'COMPILE FROM ',0);
DECLARE BEGIN(*) BYTE DATA (0DH,'RUNNING',0);
DECLARE FOUND(*) BYTE DATA (' ERROR(S) FOUND',0);
DECLARE NOERR(*) BYTE DATA (0DH,'NO',0);
DECLARE FINISH(*) BYTE DATA (0DH,'COMPILATION FINISHED',0);
DECLARE NOFIL(*) BYTE DATA (0DH,'EMPTY FILE WAS GIVEN',0);
DECLARE GOON(*) BYTE DATA (0DH,'PRESS (Y) TO EXECUTE THE PROGRAM',0);
DECLARE ANY(*) BYTE DATA (0DH,'PRESS ANY KEY TO EXIT',0);

BINASC: PROCEDURE(NUMBER) EXTERNAL;
  DECLARE NUMBER BYTE;
  END BINASC;

CRTOUT: PROCEDURE(MSG$ADR) EXTERNAL;
  DECLARE MSG$ADR ADDRESS;
  END CRTOUT;

FILZER: PROCEDURE(STR$PTR,CNT) EXTERNAL;
  DECLARE STR$PTR ADDRESS;
  DECLARE CNT ADDRESS;
  END FILZER;
```

```
MOVBD: PROCEDURE(SRC$ADR,LENGTH,DES$ADR) EXTERNAL;
  DECLARE (SRC$ADR,DES$ADR) ADDRESS;
  DECLARE LENGTH BYTE;
  END MOVBD;

CHK$LINE: PROCEDURE (LNO) EXTERNAL;
  DECLARE LNO BYTE;
  END CHK$LINE;

EMPTY: PROCEDURE(STRTADD) BYTE EXTERNAL;
  DECLARE STRTADD ADDRESS;
  END EMPTY;

GETCH: PROCEDURE BYTE EXTERNAL;
  END GETCH;

INTERPRET: PROCEDURE(LN) EXTERNAL;
  DECLARE LN BYTE;
  END INTERPRET;

ERRMSG: PROCEDURE(TYP) EXTERNAL;
  DECLARE TYP BYTE;
  END ERRMSG;

LSS002: PROCEDURE EXTERNAL;
  END LSS002;

LSS003: PROCEDURE EXTERNAL;
  END LSS003;

LSS004: PROCEDURE EXTERNAL;
  END LSS004;

LSS005: PROCEDURE EXTERNAL;
  END LSS005;

/*********************************************
* STATE    : CPS001
* FUNCTION :
* ENTERS   : CPSCAN
* EXITS    : CPS002
*
*/
CPS001: BEGIN$STATE

  CALL CRTOUT(.COMP);
  FIRST=0;
  LAST=0;
  RI=0;
  KEY=0;
  REDSTA=4;
  EDSTA=2;
  RETURN;
END CPS001;

/*********************************************
* STATE    : CPS006
* FUNCTION : COMPILATION STATE
* ENTERS   : CPS005
* EXITS    :
```

```
*
*/
CPS006: BEGIN$STATE

CALL CRT$OUT(.BEGIN);
IDLCTR=0;
KEY=0;
ENDFLG=0;

DO LINE$NO=FIRST TO LAST;
  IF EMPTY(.LINE(LINE$NO).LCHAR(0)) <> 0 THEN
    DO;
      KEY=1;
      CALL CHK$LINE(LINE$NO);
    END;
END;

IF ENDFLG = 0 THEN
  DO;
    LINE$NO=LINE$NO-1;
    CALL ERRMSG(7);
  END;

IF KEY <> 0 THEN
  DO;
    IF IDLCTR=0 THEN
      DO;
        CALL CRT$OUT(.NOERR);
        ENDFLG=0;
        EDSTA=7;
      END;
    ELSE
      DO;
        CALL CRT$OUT(.CR);
        CALL BINASC(IDLCTR);
        CALL CRT$OUT(.CONV(0));
        EDSTA=0;
      END;
    CALL CRT$OUT(.FOUND);
  END;
ELSE
  DO;
    CALL CRT$OUT(.NOFIL);
    EDSTA=0;
  END;
CALL CRT$OUT(.CR);
CALL CRT$OUT(.FINISH);
IDLCTR=0;
KEY=0;
RETURN;

END CPS006;

/***********************************************
*  STATE    : CPS007
*  FUNCTION : PREPRATION STATE OF INTERPRETER
*  ENTERS   : CPS006
*  EXITS    : CPS008,CPS000
*
*/
```

```
CPS007: BEGIN$STATE

CALL CRT$OUT(.GOON);
CALL CRT$OUT(.ANY);
CALL CRT$OUT(.CR);
KEY=GET$KEY;

IF KEY='Y' THEN
  EDSTA=8;
ELSE
  EDSTA=0;
RETURN;

END CPS007;

/*********************************************
*  STATE    : CPS008
*  FUNCTION : INTERPRETATION STATE
*  ENTERS   : CPS007
*  EXITS    :
*
*/
CPS008: BEGIN$STATE

LINE$NO=FIRST;
DO WHILE LINE$NO <= LAST;
  IF EMPTY(.LINE(LINE$NO).LCHAR(0)) <> 0 THEN
    DO;
      CALL INTERPRET(LINE$NO);
      IF ENDFLG <> 0 THEN
        DO;
          ENDFLG=0;
          JMPFLG=0;
          EDSTA=0;
          RETURN;
        END;
    END;
  LINE$NO=LINE$NO+1;
END;
EDSTA=0;

RETURN;

END CPS008;

/*****************************************************
*  PROCEDURE : CPSCAN
*  FUNCTION  : ROUTES THE LISTING STATES
*  PLM CALL  : NONE
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : CPSCAN
*  CALLS     : ALL CPS...
*/
CPSCAN: PROCEDURE PUBLIC;

  DO CASE EDSTA;
    RETURN;
    CALL CPS001;
    CALL LSS002;
```

```
    CALL LSS003;
    CALL LSS004;
    CALL LSS005;
    CALL CPS006;
    CALL CPS007;
    CALL CPS008;
  END;
END CPSCAN;

/*****************************************************
*  PROCEDURE : CPTCMD
*  FUNCTION  : EXECUTIVER OF COMPILER
*  PLM CALL  :
*  INPUT     :
*  OUTPUT    :
*  GLOBALS   :
*  CALLS     :
*/

CPTCMD: PROCEDURE PUBLIC;

  EDSTA=1;
CPLOOP:
  CALL CPSCAN;
  IF EDSTA <> 0 THEN
    GOTO CPLOOP;
  RETURN;
END CPTCMD;

/*********************************************/

END COMP1;
```

```
$PAGEWIDTH(132)
/*************************************************
*                                                *
*         UNIVERSAL MEASUREMENT SYSTEM           *
*             CONTROLLER WITH GPIB               *
*                                                *
*                    COMPILER                    *
*                                                *
*  FILE NAME   : CP002P.SO9                      *
*  MODULE DEF. : COMPILATION SUPPORT ROUTINES    *
*  DATE        : JULY 31 1986                    *
*  AUTHOR      : ILHAN NEDIM ALP                 *
*                                                *
*************************************************/

COMP2:
DO;
DECLARE (EDSTA,
        COLUMN,
        LINE$NO,
        KEY,
        IDLCTR,
        REDSTA,
        RI,
        FIRST,
        LAST,
        ENDFLG,
        JMPFLG) BYTE EXTERNAL;

DECLARE LINE(1) STRUCTURE(
        LCHAR(32)       BYTE) EXTERNAL;

DECLARE CONV(6) BYTE EXTERNAL;

DECLARE DISBUF(33) BYTE EXTERNAL;

DECLARE PARTBL(*) BYTE DATA (15,2,
        'I',1,'D',1,'W',0,'=',2,'+',3,
        '-',3,')',4,'P',5,'E',4,'C',6,
        'J',0,'R',4,'S',4,'B',5,'D',5);

DECLARE ERRMES0(*) BYTE DATA (0DH,'SYNTAX ERROR IN ',0);
DECLARE ERRMES1(*) BYTE DATA (0DH,'UNRESOLVED INSTRUCTION IN ',0);
DECLARE ERRMES2(*) BYTE DATA (0DH,'WRONG OPERAND IN ',0);
DECLARE ERRMES3(*) BYTE DATA (0DH,'UNEXISTENT VARIABLE IN ',0);
DECLARE ERRMES4(*) BYTE DATA (0DH,'UNTERMINATED INSTRUCTION IN ',0);
DECLARE ERRMES5(*) BYTE DATA (0DH,'MISSING QUOTATION IN ',0);
DECLARE ERRMES6(*) BYTE DATA (0DH,'INVALID CHARACTER IN ',0);
DECLARE ERRMES7(*) BYTE DATA (0DH,'END NOT FOUND IN ',0);
DECLARE ERRMES8(*) BYTE DATA (0DH,'OUT OF RANGE NUMBER IN ',0);
DECLARE ERRMES9(*) BYTE DATA (0DH,'WRONG DATA TYPE IN ',0);

DECLARE NEAR(*) BYTE DATA (' NEAR ',0);

BINASC: PROCEDURE(NUMBER) EXTERNAL;
 DECLARE NUMBER BYTE;
 END BINASC;

PRTOUT: PROCEDURE(MSG$ADR) EXTERNAL;
 DECLARE MSG$ADR ADDRESS;
```

```
  END CRTOUT;

CHECK$KEY: PROCEDURE(KEYNO) BYTE EXTERNAL;
  DECLARE KEYNO BYTE;
  END CHECK$KEY;

FILZER: PROCEDURE(STR$PTR,CNT) EXTERNAL;
  DECLARE STR$PTR ADDRESS;
  DECLARE CNT ADDRESS;
  END FILZER;

MOVBD: PROCEDURE(SRC$ADR,LENGTH,DES$ADR) EXTERNAL;
  DECLARE (SRC$ADR,DES$ADR) ADDRESS;
  DECLARE LENGTH BYTE;
  END MOVBD;

/*****************************************************
*  PROCEDURE : ERRMSG
*  FUNCTION  : DISPLAY COMPILATION ERROR MESSAGES
*  PLM CALL  :  CALL ERRMSG(ERR$TYP)
*  INPUT     : ERR$TYP.. DISPLAYED ERROR TYPE
*  OUTPUT    : NONE
*  GLOBALS   : CONV
*  CALLS     : BINASC,CRTOUT
*/

ERRMSG: PROCEDURE(TYP) PUBLIC;

  DECLARE TYP BYTE;

  DO CASE TYP;
    CALL CRT$OUT(.ERRMES0);
    CALL CRT$OUT(.ERRMES1);
    CALL CRT$OUT(.ERRMES2);
    CALL CRT$OUT(.ERRMES3);
    CALL CRT$OUT(.ERRMES4);
    CALL CRT$OUT(.ERRMES5);
    CALL CRT$OUT(.ERRMES6);
    CALL CRT$OUT(.ERRMES7);
    CALL CRT$OUT(.ERRMES8);
    CALL CRT$OUT(.ERRMES9);
  END;
  CALL BIN$ASC(LINE$NO);
  CALL CRT$OUT(.CONV(0));
  CALL CRT$OUT(.NEAR);
  CALL CRT$OUT(.DISBUF(0));
  IDLCTR=IDLCTR+1;
  RETURN;

END ERRMSG;

/*****************************************************
*  PROCEDURE : CHK$DATA
*  FUNCTION  : CONTROLS THE DATA FIELD
*  PLM CALL  :  CALL CHK$DATA
*  INPUT     : NONE
*  OUTPUT    : ERROR MESSAGE TO CRT
*  GLOBALS   : DISBUF
*  CALLS     : NONE
*/
```

```
CHK$DATA: PROCEDURE PUBLIC;

DECLARE I BYTE;

IF DISBUF(3) <> 22H THEN
  CALL ERRMSG(5);
ELSE
  DO;
    DO I=4 TO 31;
      IF DISBUF(I) <> 0 THEN
        DO;
          IF DISBUF(I) = 22H THEN
            DO;
              IF DISBUF(I+1) <> 0 THEN
                CALL ERRMSG(6);
              RETURN;
            END;
        END;
      ELSE
        DO;
          CALL ERRMSG(5);
          RETURN;
        END;
    END;
    CALL ERRMSG(5);
  END;

END CHK$DATA;

/*****************************************************
*  PROCEDURE : CHK$DIG
*  FUNCTION  : CHECKS THE RANGE OF DIGIT OPERANDS
*  PLM CALL  :  CALL CHK$DIG(TYP)
*  INPUT     : TYP.. DIGIT OPERAND TYPE
*  OUTPUT    : NONE
*  GLOBALS   : CONV
*  CALLS     : BINASC,CRTOUT
*/

CHK$DIG: PROCEDURE(YP) PUBLIC;

  DECLARE (YP,DIGIT) BYTE;

  DIGIT= (DISBUF(1) AND 03H)*10+(DISBUF(2) AND 03H);

  DO CASE YP;
  DO;
    IF DIGIT >= 32 THEN
      CALL ERR$MSG(8);
  END;
  DO;
    IF (DIGIT > LAST) OR (DIGIT < FIRST) THEN
      CALL ERR$MSG(8);
  END;
  END; /* END OF DO CASE */
  RETURN;

END CHK$DIG;
```

```
/*****************************************************
*  PROCEDURE : VALID$DIG
*  FUNCTION  : CHECKS THE DATA AREA WHETHER THEY ARE
*              DIGITS OR NOT
*  PLM CALL  :  CALL VALID$DIG
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : NONE
*  CALLS     : CHK$KEY,ERRMSG
*/

VALID$DIG: PROCEDURE PUBLIC;

DECLARE (DIG,K) BYTE;

  DO K=4 TO 29;
    DIG=DISBUF(K);
    IF DIG <> 0 THEN
      DO;
        IF DIG <> 22H THEN
          DO;
            IF CHECK$KEY(DIG) <> 6 THEN
              DO;
                CALL ERRMSG(9);
                RETURN;
              END;
          END;
        RETURN;
      END;
    RETURN;
  END;
END VALID$DIG;

/*****************************************************
*  PROCEDURE : CHK$TYP
*  FUNCTION  : CHECKS THE TYPE OF OPERAND
*  PLM CALL  :  CALL CHK$TYP(TYPE)
*  INPUT     : TYPE.. TYPE OF OPERAND
*  OUTPUT    : ERROR MESSAGE IF IMPROPER TYPE EXISTS
*  GLOBALS   : NONE
*  CALLS     : ERRMSG, CHK$DATA, CHK$DIG
*/

CHK$TYP: PROCEDURE(TYPE);

 DECLARE (OP1,OP2,TYPE,I) BYTE;

 OP1=CHECK$KEY(DISBUF(1));
 OP2=CHECK$KEY(DISBUF(2));

 DO CASE TYPE;
   /* Type..0  W,J */
 DO;
   IF OP1 <> 6 OR OP2 <> 6 THEN
     CALL ERRMSG(2);
   ELSE
     IF DISBUF(3) <> 0 THEN
       CALL ERRMSG(6);
   IF DISBUF(0) = 'J' THEN
     CALL CHK$DIG(1);
```

```
END;
  /* Type..1  I,O */
DO;
  IF OP1 <> 6 OR OP2 <> 6 THEN
    CALL ERRMSG(2);
  ELSE
    CALL CHK$DATA;
  CALL CHK$DIG(0);
  IF DISBUF(0) = 'I' THEN
    DO;
      IF DISBUF(4) <85 OR DISBUF(4) >90 THEN
        CALL ERRMSG(3);
    END;
END;
  /* Type..2  = */
DO;
  IF OP1 <> 4 OR OP2 <> 5 THEN
    CALL ERRMSG(2);
  ELSE
    DO;
      IF DISBUF(2) <> 'K' AND
         DISBUF(2) <> 'L' THEN
        CALL ERRMSG(3);
      ELSE
        CALL CHK$DATA;
      CALL VALID$DIG;
    END;
END;
  /* Type..3  +,- */
DO;
  IF OP1 <> 4 OR OP2 <> 5 THEN
    CALL ERRMSG(2);
  ELSE
    DO;
      IF DISBUF(2) <> 'K' AND
         DISBUF(2) <> 'L' THEN
        CALL ERRMSG(3);
      ELSE
        IF DISBUF(3) <> 0 THEN
          CALL ERRMSG(6);
    END;
END;
  /* Type..4  ),E,R,S */
DO;
  IF OP1 <> 4 OR OP2 <> 4 THEN
    CALL ERRMSG(2);
  ELSE
    DO;
      IF (DISBUF(3) <> 0) AND (DISBUF(0) <> 'R') THEN
        CALL ERRMSG(6);
      IF DISBUF(0) = 'E' THEN
        ENDFLG=1;
    END;
END;
  /* Type..5  P,D,B */
DO;
  IF OP1 <> 4 OR OP2 <> 5 THEN
    CALL ERRMSG(2);
  ELSE
    DO;
```

```
      IF DISBUF(2) < 85 OR DISBUF(2) > 90 THEN
        CALL ERRMSG(3);
      IF DISBUF(3) <> 0 THEN
        DO;
          IF DISBUF(3) <> 22H THEN
            CALL ERRMSG(5);
          ELSE
            CALL CHK$DATA;
        END;
    END;
END;
  /* Type..6 . C */
DO;
  IF OP1 <> 5 OR OP2 <> 5 THEN
    CALL ERRMSG(2);
  ELSE
    DO;
      IF (DISBUF(1) <> 'K' AND DISBUF(1) <> 'L') OR
        (DISBUF(2) <> 'K' AND DISBUF(2) <> 'L') THEN
        CALL ERRMSG(3);
      ELSE
        IF DISBUF(3) <> 0 THEN
          CALL ERRMSG(6);
    END;
END;
END; /* END OF DO CASE */

RETURN;

END CHK$TYP;

/*****************************************************
*  PROCEDURE : PARSING
*  FUNCTION  : PARSING THE INSTRUCTION
*  PLM CALL  : CALL PARSING
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : DISBUF
*  CALLS     : CHK$TYP
*/

PARSING: PROCEDURE PUBLIC;
  DECLARE (RECCNT,ENTCNT,I,K,TP) BYTE;

  RECCNT=PARTBL(0);
  ENTCNT=PARTBL(1);
  K=2;
  DO I=0 TO RECCNT;
    IF DISBUF(0) = PARTBL(K) THEN
      DO;
        TP=PARTBL(K+1);
        CALL CHK$TYP(TP);
        RETURN;
      END;
    ELSE
      K=K+ENTCNT;
  END;
  CALL ERRMSG(1);
  RETURN;
```

```
END PARSING;

/*************************************************
*  PROCEDURE : CHK$LINE
*  FUNCTION  : CHECKS THE GIVEN LINE AND SEPERATES
*              THE INSTRUCTIONS WHICH ARE GIVEN THE
*              SAME LINE
*  PLM CALL  :  CALL CHK$LINE(LNO);
*   INPUT    : LNO...CHECKED LINE NO
*   OUTPUT   : NONE
*   GLOBALS  : NONE
*   CALLS    : PARSING, FILZER
*/

CHK$LINE: PROCEDURE(LNO) PUBLIC;

 DECLARE (LNO,M,N,D) BYTE;

  CALL FILZER(.DISBUF(0),33);
  M=0;
  N=0;
BEGIN: DO WHILE M <= 31;

   D=LINE(LNO).LCHAR(M);
   IF D <> 0 THEN
     DO;
       IF D <> ':' THEN
         DO;
           IF M=31 THEN
             DO;
               CALL ERRMSG(4);
               RETURN;
             END;
           DISBUF(N)=LINE(LNO).LCHAR(M);
           M=M+1;
           N=N+1;
           GO TO BEGIN;
         END;
       ELSE
         DO;
           IF M=0 THEN
             DO;
               CALL MOVBD(.LINE(LNO).LCHAR(0),32,.DISBUF(0));
               CALL ERRMSG(0);
               RETURN;
             END;
           CALL PARSING;
           N=0;
           M=M+1;
           CALL FILZER(.DISBUF(0),33);
           GO TO BEGIN;
         END;
     END;
   ELSE
     DO;
       IF DISBUF(0) <> 0 THEN
         DO;
           CALL ERRMSG(4);
           CALL PARSING;
         END;
```

```
        RETURN;
      END;

  END;

  RETURN;

  END CHK$LINE;

  END COMP2;
```

```
$PAGEWIDTH(132)
/*****************************************************
**                                                  **
**         UNIVERSAL MEASUREMENT SYSTEM             **
**             CONTROLLER WITH GPIB                 **
**                                                  **
**                  INTERPRETER                     **
**                                                  **
**  FILE NAME   : CP003P.S09                        **
**  MODULE DEF. : INTERPRETER ROUTINES              **
**  DATE        : AUGUST 05 1986                    **
**  AUTHOR      : ILHAN NEDIM ALP                   **
**                                                  **
*****************************************************/

COMP3:
DO;
DECLARE (EDSTA,
        COLUMN,
        LINE$NO,
        KEY,
        IDLCTR,
        REDSTA,
        RI,
        FIRST,
        LAST,
        ENDFLG,
        JMPFLG) BYTE EXTERNAL;

DECLARE GETKEY  LITERALLY 'GETCH';

DECLARE MULTBL(*) ADDRESS DATA (1,10,100,1000);

DECLARE EXECTBL(*) BYTE DATA (15,2,
                  'I',0,'D',1,'W',2,'=',3,'+',4,
                  '-',5,')',6,'P',7,'E',8,'C',9,
                  'J',10,'S',11,'B',12,'D',13,'R',14);

DECLARE CR(*) BYTE DATA (0DH,0);
DECLARE SP(*) BYTE DATA (20H,0);
DECLARE BEEP(*) BYTE DATA (07H,0);
DECLARE C(*) BYTE DATA (0DH,'PRESS (C) TO CONTINUE',0);
DECLARE R(*) BYTE DATA (0DH,'PRESS (R) TO RESTART',0);
DECLARE E(*) BYTE DATA (0DH,'PRESS (E) TO EXIT',0);
DECLARE PRINTERFAIL(*) BYTE DATA (07H,0DH,'PRINTER FAIL',0);

DECLARE LINE(1) STRUCTURE(
        LCHAR(32)       BYTE) EXTERNAL;

DECLARE STAFL(6) BYTE EXTERNAL;
DECLARE LPCNT BYTE EXTERNAL;

DECLARE OBUFER(1) STRUCTURE(
        OCHR(80)        BYTE) EXTERNAL;

DECLARE DISBUF(33) BYTE EXTERNAL;

DECLARE INBUF(32) BYTE EXTERNAL;

DECLARE REG(1) STRUCTURE(
```

```
        CHR(32)         BYTE) EXTERNAL;

DECLARE OBUF(28) BYTE EXTERNAL;

DECLARE LLIST(2) BYTE EXTERNAL;

DECLARE TLIST(2) BYTE EXTERNAL;

DECLARE (VARA,VARB) ADDRESS EXTERNAL;

DECLARE OBFCNT BYTE EXTERNAL;

EXRECV: PROCEDURE EXTERNAL;
  END EXRECV;

EXSEND: PROCEDURE EXTERNAL;
  END EXSEND;

FILZER: PROCEDURE(STR$PTR,CNT) EXTERNAL;
  DECLARE STR$PTR ADDRESS;
  DECLARE CNT ADDRESS;
  END FILZER;

EMPTY: PROCEDURE(STRTADD) BYTE EXTERNAL;
  DECLARE STRTADD ADDRESS;
  END EMPTY;

WRWAIT: PROCEDURE EXTERNAL;
  END WRWAIT;

CRTOUT: PROCEDURE(MSG$ADR) EXTERNAL;
  DECLARE MSG$ADR ADDRESS;
  END CRTOUT;

MOVBD: PROCEDURE(SRC$ADR,LENGTH,DES$ADR) EXTERNAL;
  DECLARE (SRC$ADR,DES$ADR) ADDRESS;
  DECLARE LENGTH BYTE;
  END MOVBD;

GETCH: PROCEDURE BYTE EXTERNAL;
  END GETCH;

CONIN: PROCEDURE BYTE EXTERNAL;
  END CONIN;

/*****************************************************
*  PROCEDURE : GETLP
*  FUNCTION  : SEARCH AN EMPTY PRINTER BUFFER AND
*              RETURNS ITS NUMBER
*  PLM CALL  :  J = GETLP;
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : STAFL
*  CALLS     : NONE
*/

GETLP: PROCEDURE BYTE PUBLIC;
  DECLARE K BYTE;

  K=5;
```

```
  IF STAFL(LPCNT)=0 THEN
    DO;
      STAFL(LPCNT)=1;
      K=LPCNT;
      LPCNT=LPCNT+1;
      IF LPCNT > 5 THEN
        LPCNT=0;
    END;
  RETURN K;

END GETLP;

/*****************************************************
*  PROCEDURE : SEARCH$QUOT
*  FUNCTION  : SEARCHES THE SECOND QUOTATION OF DATA
*              FIELD AND ZEROS IT AND ALSO RETURNS
*              THE NUMBER OF CHARACTER
*  PLM CALL  : J=SEARCH$QUOT
*  INPUT     : NONE
*  OUTPUT    : NUMBER OF CHARACTER
*  GLOBALS   : DISBUF
*  CALLS     : NONE
*/

SEARCH$QUOT: PROCEDURE BYTE PUBLIC;
  DECLARE PT BYTE;

  PT=4;
  AGAIN: IF DISBUF(PT) = 22H THEN
           DO;
             DISBUF(PT)=0;
             RETURN PT;
           END;
         ELSE
           DO;
             PT=PT+1;
             GO TO AGAIN;
           END;

END SEARCH$QUOT;

/*****************************************************
*  PROCEDURE: CONVASCBIN
*  FUNCTION : CONVERTS THE TWO BYTE ASCII DIGITS INTO
*             THE ONE BYTE BINARY
*  PLM CALL :  J = CONVASCBIN
*  INPUTS   : NONE
*  OUTPUTS  : ONE BYTE BINARY VALUE
*  GLOBALS  : DISBUF
*  CALLS    : NONE
*/

CONVASCBIN: PROCEDURE BYTE PUBLIC;
  DECLARE RESULT BYTE;

  DISBUF(1)=DISBUF(1) AND 0FH;
  DISBUF(2)=DISBUF(2) AND 0FH;
  RESULT=DISBUF(1)*10 + DISBUF(2);
  RETURN RESULT;
```

```
OBUF(OBFCNT)=0Dh;
OBUF(OBFCNT+1)=0Ah;
OBFCNT=OBFCNT+2;
CALL EXSEND;
RETURN;
END OINS;

/*************************************************
* PROCEDURE : WINS
* FUNCTION  : WAIT INSTRUCTION
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

WINS: PROCEDURE PUBLIC;

DECLARE (W,WW) BYTE;

  DO W=0 TO CONVASCBIN;
    DO WW=0 TO 24;
      CALL WAWAIT;
    END;
  END;
  RETURN;

END WINS;

/*************************************************
* PROCEDURE : DCRINS
* FUNCTION  : DECREMENTS THE CONTENT OF VARIABLE
* PLM CALL  : NONE
* INPUT     : NONE
* OUTPUT    : NONE
* GLOBALS   :
* CALLS     :
*/

DCRINS: PROCEDURE PUBLIC;

  IF DISBUF(2) = 'K' THEN
    VARA=VARA-1;
  IF DISBUF(2) = 'L' THEN
    VARB=VARB-1;

RETURN;

END DCRINS;

/*************************************************
* PROCEDURE : INCINS
* FUNCTION  : INCREMENTS THE CONTENT OF THE VARIABLE
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
```

```
END CONVASCBIN;

/*****************************************************
*  PROCEDURE : IINS
*  FUNCTION  : REALIZE (I) INSTRUCTION
*  PLM CALL  : NONE
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : DISBUF,TLIST,INBUF
*  CALLS     : EXRECV,FILZER,ASCBIN
*/

IINS: PROCEDURE PUBLIC;
  DECLARE N BYTE;

 TLIST(0)=CONVASCBIN OR 40H;
 TLIST(1)=0FFH;
 CALL FILZER(.INBUF(0),32);
 CALL EXRECV;
 N=DISBUF(4)-85;
 CALL MOVBD(.INBUF(0),32,.REG(N).CHR(0));

 RETURN;

END IINS;

/*****************************************************
*  PROCEDURE : DINS
*  FUNCTION  : REALIZE (D) INSTRUCTION
*  PLM CALL  : NONE
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : DISBUF,LLIST,OBUF
*  CALLS     : FILZER,ASCBIN
*/

DINS: PROCEDURE PUBLIC;

DECLARE (D,L) BYTE;

LLIST(0)=CONVASCBIN OR 20H;
LLIST(1)=0FFH;
CALL FILZER(.OBUF(0),28);
OBFCNT=0;

DO L=4 TO 29;
  D=DISBUF(L);
  IF D <> 22H THEN
    DO;
      IF D <> ',' THEN
        DO;
          OBUF(OBFCNT)=D;
          OBFCNT=OBFCNT+1;
        END;
    END;
  ELSE
    GO TO GOON;
END;

GOON:
```

```
 */

INCINS: PROCEDURE PUBLIC;

  IF DISBUF(2) = 'K' THEN
    VARA=VARA+1;
  IF DISBUF(2) = 'L' THEN
    VARB=VARB+1;
  RETURN;

END INCINS;

/*****************************************************
* PROCEDURE : EQINS
* FUNCTION  : REALIZE (=) INSTRUCTION
* PLM CALL  : NONE
* INPUT     : NONE
* OUTPUT    : NONE
* GLOBALS   :
* CALLS     :
*/

EQINS: PROCEDURE PUBLIC;

DECLARE (M,I) BYTE,
        (VDUM,VALUE) ADDRESS;

M=4;
I=0;

DO WHILE DISBUF(M) <> 22H;
  DISBUF(M)=DISBUF(M) AND 0FH;
  M=M+1;
  I=I+1;
END;

IF I > 4 THEN
  I=4;
VALUE=0;

DO M=1 TO I;
  VDUM=DOUBLE(DISBUF(M+3));
  VALUE=VALUE+VDUM*MULTBL(I-M);
END;

IF DISBUF(2) = 'K' THEN
  VARA= VALUE;
IF DISBUF(2) = 'L' THEN
  VARB= VALUE;
RETURN;

END EQINS;

/*****************************************************
* PROCEDURE : JMPINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
```

```
* CALLS     :
*/

JMPINS: PROCEDURE PUBLIC;

  LINE$NO=CONVASCBIN - 1;
  JMPFLG=1;
  RETURN;

END JMPINS;
/*****************************************************
* PROCEDURE : ENDINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

ENDINS: PROCEDURE PUBLIC;

  ENDFLG=1;
  JMPFLG=1;
  RETURN;

END ENDINS;

/*****************************************************
* PROCEDURE : CMPINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

CMPINS: PROCEDURE PUBLIC;

  IF DISBUF(1) = 'K' THEN
    DO;
      IF VARA ) VARB THEN RETURN;
      JMPFLG=1;
    END;
  ELSE
    DO;
      IF VARB ) VARA THEN RETURN;
      JMPFLG=1;
    END;

RETURN;

END CMPINS;

/*****************************************************
* PROCEDURE : REMINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
```

```
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

REMINS: PROCEDURE PUBLIC;

  JMPFLG=1;

  RETURN;

END REMINS;

/*****************************************************
* PROCEDURE : STPINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

STPINS: PROCEDURE PUBLIC;
 DECLARE I BYTE;
 DO I=0 TO 9;
  CALL CRTOUT(.BEEP);
 END;
  CALL CRTOUT(.C);
  CALL CRTOUT(.R);
  CALL CRTOUT(.E);
  CALL CRTOUT(.CR);

  KEY=GETKEY;

  IF KEY='C' THEN
    RETURN;
  IF KEY='R' THEN
    DO;
     LINEND=FIRST-1;
     JMPFLG=1;
    END;
  IF KEY='E' THEN
    CALL ENDINS;

  RETURN;

END STPINS;

/*****************************************************
* PROCEDURE : DINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

DINS: PROCEDURE PUBLIC;
```

```
  DECLARE (I,J) BYTE;

  CALL CRTOUT(.CR);
  IF DISBUF(3) <> 0 THEN
    J=SEARCH$QUOT;
  CALL CRTOUT(.DISBUF(4));
  CALL CRTOUT(.SP);
  I=DISBUF(2)-85;
  CALL CRTOUT(.REG(I).CHR(0));
  RETURN;

END DINS;

/*************************************************
* PROCEDURE : BINS
* FUNCTION  :
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

BINS: PROCEDURE PUBLIC;

  CALL CRTOUT(.BEEP);
  CALL DINS;
  RETURN;

END BINS;

/*************************************************
* PROCEDURE : PINS
* FUNCTION  : PRINTS THE DATA FIELD IF EXISTS AND
*             CONTENTS OF THE SPECIFIED REGISTER
* PLM CALL  :
* INPUT     :
* OUTPUT    :
* GLOBALS   :
* CALLS     :
*/

PINS: PROCEDURE PUBLIC;
  DECLARE (K,J,M,I,L,P) BYTE;

  K=SEARCH$QUOT;
  J=GETLP;
  M=DISBUF(2)-85;
  IF J >= 6 THEN
    DO;
      CALL CRTOUT(.PRINTERFAIL);
      RETURN;
    END;
  DO I=0 TO 79;
    OBUFER(J).OCHR(I)=20H;
  END;

  P=0;
  IF K <> 4 THEN
    DO;
```

```
      L=K-4;
      CALL MOVBD(.DISBUF(4),L,.OBUFER(J).OCHR(P));
      P=L-1;
    END;
  CALL MOVBD(.REG(M).CHR(0),32,.OBUFER(J).OCHR(P));
  STAFL(J)=2;
  RETURN;

END PINS;

/*****************************************************
*  PROCEDURE : PINBUF
*  FUNCTION  : DISPLAYS THE INBUF TO CRT
*  PLM CALL  :
*  INPUT     :
*  OUTPUT    :
*  GLOBALS   :
*  CALLS     :
*/

PINBUF: PROCEDURE PUBLIC;

  CALL CRTOUT(.CR);
  CALL CRTOUT(.INBUF(0));
  RETURN;

END PINBUF;

/*****************************************************
*  PROCEDURE : EXECUTIVE
*  FUNCTION  : EXECUTES THE INSTRUCTIONS
*  PLM CALL  : CALL EXECUTIVE
*  INPUT     : NONE
*  OUTPUT    : NONE
*  GLOBALS   : DISBUF
*  CALLS     : ALL INSTRUCTION SUBROUTINES
*/

EXECUTIVE: PROCEDURE PUBLIC;
  DECLARE (RECCNT,ENTCNT,K,I,PARAM) BYTE;

RECCNT=EXECTBL(0);
ENTCNT=EXECTBL(1);
K=2;
DO I=0 TO RECCNT;
  IF DISBUF(0)=EXECTBL(K) THEN
    DO;
      PARAM=EXECTBL(K+1);
      GO TO FIND;
    END;
  ELSE
    K=K+ENTCNT;
END;
FIND:
IF PARAM >= RECCNT THEN
  RETURN;

DO CASE PARAM;
  CALL IINS;
  CALL OINS;
```

```
    CALL WINS;
    CALL EBINS;
    CALL INCINS;
    CALL DCRINS;
    CALL PINBUF;
    CALL PINS;
    CALL ENDINS;
    CALL CMPINS;
    CALL JMPINS;
    CALL STPINS;
    CALL BINS;
    CALL DINS;
    CALL REMINS;
  END; /* END OF DO CASE */

  RETURN;

  END EXECUTIVE;

  /*****************************************************
  *  PROCEDURE : CHK$USRT
  *  FUNCTION  : CHECKS THE MONITOR USART
  *  PLM CALL  : CALL CHK$USRT
  *  INPUT     : NONE
  *  OUTPUT    : NONE
  *  GLOBALS   : NONE
  *  CALLS     : CONIN
  */

  CHK$USRT: PROCEDURE PUBLIC;
    DECLARE M BYTE;

    M=CONIN;
    IF M = 0 THEN RETURN;
    CALL ENDINS;
    RETURN;

  END CHK$USRT;

  /*****************************************************
  *  PROCEDURE : INTERPRET
  *  FUNCTION  : REALIZE THE INTERPRATATION OF
  *              INSTRUCTIONS
  *  PLM CALL  :  CALL INTERPRET(LN);
  *  INPUT     : LN....LINE NUMBER
  *  OUTPUT    : NONE
  *  GLOBALS   : DISBUF, LINE
  *  CALLS     : EXEC
  */

  INTERPRET: PROCEDURE(LNO) PUBLIC;
    DECLARE (LNO,M,N,D) BYTE;

  CALL FILZER(.DISBUF(0),33);
  M=0;
  N=0;
  START: DO WHILE M <= 31;
    D=LINE(LNO).LCHAR(M);
    IF D <> 0 THEN
      DO;
```

```
      IF D () ';' THEN
        DO;
          DISBUF(N)=D;
          M=M+1;
          N=N+1;
          GOTO START;
        END;
      ELSE
        DO;
          JMPFLG=0;
          CALL EXECUTIVE;
          CALL CHK$USRT;
          IF JMPFLG () 0 THEN
            RETURN;
          N=0;
          M=M+1;
          CALL FILZER(.DISBUF(0),33);
          GOTO START;
        END;
    END;
  RETURN;
  END;

RETURN;

END INTERPRET;

/*****************************************/

END COMP3;
```

```
$PAGEWIDTH(120)
;*****************************************************
;*                                                   **
;*          UNIVERSAL MEASUREMENT SYSTEM             **
;*             CONTROLLER WITH GPIB                  **
;*                                                   **
;*                GPIB  ROUTINES                     **
;*                                                   **
;*  FILE NAME   : GP100A.S01                         **
;*  MODULE DEF. : GPIB ROUTINES                      **
;*  DATE        : APRIL 13 1986                      **
;*  AUTHOR      : ILHAN NEDIM ALP                    **
;*                                                   **
;*****************************************************
;
        NAME    GPIB
;
$INCLUDE(:F0:GPIB.ASM)
;
        PUBLIC  INIT
        PUBLIC  SEND
        PUBLIC  RECV
        PUBLIC  REME
;
        CSEG
;
;*****************************************************
;  PROCEDURE : INIT
;  FUNCTION  : INITIALIZATION ROUTINE
;  PLM CALL  : None
;  INPUT     : None
;  OUTPUT    : None
;  GLOBALS   : None
;  CALLS     : None
;  DESTROYS  : A,F
;
INIT:   MVI     A,INTM          ;Enable TCI
        OUT     INTMR           ;Output to 92's intr. mask reg.
        MVI     A,DTDL1         ;Disable major talker/listener
        OUT     ADR01
        MVI     A,DTDL2         ;Disable minor talker/listener
        OUT     ADR01
        MVI     A,TON           ;Talk only mode
        OUT     ADRMD
        MVI     A,CLKRT         ;3 MHz for delay timer
        OUT     AUXMD
        CLRA
        OUT     INT1
        OUT     INT2            ;Disable all 91 mask bits
        OUT     AUXMD           ;Immediate execute PON
        RET
;
;*****************************************************
;  PROCEDURE : SEND
;  FUNCTION  : SEND ROUTINE
;  PLM CALL  : None
;  INPUT     : HL listener list pointer
;              DE data buffer pointer
;              C  count.. 0 will cause no data to be
;                 sent
```

```
          B  EOS character.. software detected
OUTPUT    : None
GLOBALS   : None
CALLS     : None
DESTROYS  : A, C, DE, HL, F

D:  MVI   A,UNL           ;Send universal unlisten to turn off
    OUT   DOUT            ;any previous talker
    WAITO
    MVI   A,UNT           ;Send universal untalk
    OUT   DOUT            ;to stop previous talkers
    WAITO
    MOV   A,B             ;Get EOS character
    OUT   EOSR            ;Output it to 8291A
                          ;while listener.....
D1: RANGE 20H,3EH,SEND2   ;Check next listen address
    OUT   DOUT            ;Output to GPIB
    WAITO
    INX   H               ;Increment listener lst pointer
    JMP   SEND1           ;Loop untill non valid listener
                          ;Enable 8291A ending conditions
D2: MVI   A,GTSB          ;Goto standby
    OUT   CMD92
    MVI   A,AUX9A+EOIS    ;Send EOI with EOS character
    OUT   AUXMD
    WAITX                 ;Wait for TCI to go false
    WAITT                 ;Wait for TCI on GTSB

    MVI   A,TON           ;Put 8291A to talk only mode
    OUT   ADRMD

    CLRA

    OUT   AUXMD           ;Immediate execute PON

    Delete next 3 instructions to make count 0=256

    MOV   A,C             ;Get count
    SETF
    JZ    SEND6           ;If count=0 send no data

D3: LDAX  D               ;Get data byte
    OUT   DOUT            ;Output to GPIB
    CMP   B               ;Test EOS ... this is faster
                          ;and uses less code than using
                          ;91's END or EOI bits
    JZ    SEND5           ;If character=EOS, go finish
    WAITO
    INX   D               ;Increment buffer pointer
    DCR   C               ;Decrement count
    JNZ   SEND3           ;If count <> 0, go send
    JMP   SEND6           ;Else go finish

D5: INX   D               ;for consistency
    DCR   C               ;for consistency
    WAITO
                          ;assumptions for the next subroutine are met

D6: MVI   A,TCSY          ;Take control syncronously
    OUT   CMD92
```

```
        MVI     A,AXRA          ;Reset send EOI on EOS
        OUT     AUXMD
        WAITX
        WAITT
        RET
;
;***************************************************
;  PROCEDURE : RECV
;  FUNCTION  : RECEIVE ROUTINE
;  PLM CALL  : None
;  INPUT     : HL talker pointer
;              DE data buffer pointer
;              C  count (max. buffer size) 0 implies 256
;              B  EOS character
;  OUTPUT    : Fills buffer pointed at by DE
;  GLOBALS   : None
;  CALLS     : None
;  DESTROYS  : A, BC, DE, HL, F
;
;  RETURNS   : A= 0H normal termination--EOS detected
;              A=40H Error---count overrun
;              A(40H or A)5EH Error--- bad talk address
;
RECV:   MOV     A,B             ;Get EOS character
        OUT     EOSR            ;Output it to 9i
        RANGE   40H,5EH,RECV6
                                ;valid if 40H(= talk (= 5EH
        OUT     DOUT            ;Output talker to GPIB
        INX     H               ;Incr. pointer for consistency
        WAITO
        MVI     A,UNL           ;stop other listeners
        OUT     DOUT
        WAITO
        MVI     A,MLA           ;For completeness
        OUT     DOUT
        MVI     A,AXRA+HOEND+EDEOS
        OUT     AUXMD           ;End when EOS or EOI & Holdoff
        WAITO
        MVI     A,LON           ;Listen only
        OUT     ADRMD
        CLRA                    ;Immediate XEQ PON
        OUT     AUXMD
        MVI     A,GTSB          ;Goto standby
        OUT     CMD32
        WAITX                   ;Wait for TCI=0
        WAITT                   ;Wait for TCI=1
;
RECV1:  IN      INT1            ;Get 9i Int status (END &/or BI)
        MOV     B,A             ;Save it in B for BI check later
        ANI     ENDMK           ;Check for EOS or EOI
        JNZ     RECV2           ;Yes end--- go wait for BI
        MOV     A,B             ;No retrieve status &
        ANI     BIM             ;check for BI
        JZ      RECV1           ;No, go wait for either END or BI
        IN      DIN             ;Yes, BI--- get data
        STAX    D               ;Store it in buffer
        INX     D               ;Increment buffer pointer
        DCR     C               ;Decrement counter
        JNZ     RECV1           ;If count () 0 go back & wait
        MVI     B,40h           ;Else set error indicator
```

```
        JMP     RECV5           ;And go take control
;
RECV2:  MOV     A,B             ;Retrieve status
RECV3:  ANI     BIM             ;Check for BI
        JNZ     RECV4           ;If BI then go input data
        IN      INT1            ;Else wait last BI
        JMP     RECV3           ;In loop
;
RECV4:  IN      DIN             ;Get data byte
        STAX    D               ;Store it in buffer
        INX     D               ;Increment data pointer
        DCR     C               ;Decrement count but ignore it
        MVI     B,0             ;Set normal completion indicators
;
RECV5:  MVI     A,TCSY          ;Take control synchronously
        OUT     CMD92
        WAITX                   ;Wait for TCI=0 (7 try)
        WAITT                   ;Wait for TCI=1
;
        MVI     A,AXRA          ;Pattern to clear 91 END conditions
        OUT     AUXMD
        MVI     A,TON           ;This bit pattern already in "A"
        OUT     ADRMD           ;Output TON
        MVI     A,FNHSK         ;Finish handshake
        OUT     AUXMD
        CLRA
        OUT     AUXMD           ;Immediate execute PON-Reset LON
        MOV     A,B             ;Get completion character
;
RECV6:  RET
;
;***************************************************
;  PROCEDURE : REME
;  FUNCTION  : REMOTE ENABLE ROUTINE
;  PLM CALL  : None
;  INPUT     : None
;  OUTPUT    : None
;  GLOBALS   : None
;  CALLS     : None
;  DESTROYS  : A, F
;
REME:   MVI     A,SREM
        OUT     CMD92           ;92 asserts remote enable
        WAITX                   ;Wait for TCI=0
        WAITT                   ;Wait for TCI
;
        RET
;
;***************************************************
;
        END
;
```

```
$PAGEWIDTH(80)
/*****************************************************
**                                                  **
**          UNIVERSAL MEASUREMENT SYSTEM            **
**              CONTROLLER WITH GPIB                **
**                                                  **
**                    UTILITIES                     **
**                                                  **
**  FILE NAME   : UT010P.S03                        **
**  MODULE DEF. : UTILITIES OF EDITOR               **
**  DATE        : JULY 03 1986                      **
**  AUTHOR      : ILHAN NEDIM ALP                   **
**                                                  **
*****************************************************/

UTEDIT:DO;

DECLARE CONV(3) BYTE EXTERNAL;

DECLARE I BYTE;

DECLARE CONTAB(*) BYTE DATA (100,10,1);

DECLARE ASCTAB(*) BYTE DATA(
        0,0,0,0,0,0,0,0,1,0,0,0,0,2,0,0,
        8,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,
        4,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        6,6,6,6,6,6,6,6,6,6,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,7);

/*****************************************************
*  PROCEDURE : CHECK$KEY
*  FUNCTION  : CONTROLS AND QUALIFIES THE PRESSED KEYS
*  PLM CALL  :  =CHECK$KEY(KEYNO)
*  INPUT     : KEYNO
*  OUTPUT    : RANGE VALUE OF PRESSED KEY
*  GLOBALS   : NONE
*  CALLS     : NONE
*/

CHECK$KEY: PROCEDURE(KEYNO) BYTE PUBLIC;
  DECLARE KEYNO BYTE;

  KEYNO = KEYNO AND 7FH;
  RETURN ASCTAB(KEYNO);

END CHECK$KEY;

/*****************************************************
*  PROCEDURE : BINASC
*  FUNCTION  : CONVERTS A BYTE OF BINARY NUMBER TO
*              THREE BYTE ASCII CHARACTER
*  PLM CALL  : CALL BINASC(NUMBER)
*  INPUT     : NUMBER = WILL BE CONVERTED INTO
*  OUTPUT    : ASCII FORM VALUE IN CONV(*)
*  GLOBALS   : CONV
*  CALLS     : NONE
```

```
*/

BINASC: PROCEDURE(NUMBER) PUBLIC;
  DECLARE NUMBER BYTE;

      DO I=0 TO 2;
        CONV(I)=0;
      END;

     I=0;
START: IF NUMBER >= CONTAB(I) THEN
        DO;
          CONV(I)=CONV(I)+1;
          NUMBER=NUMBER-CONTAB(I);
        END;
       ELSE
        DO;
          I=I+1;
          IF I >= 3 THEN
            DO;
              DO I=0 TO 2;
                CONV(I)=CONV(I) OR 30H;
              END;
              RETURN;
            END;
        END;
       GO TO START;

END BINASC;

END UTEDIT;
```

```
$PAGEWIDTH(80)
;*****************************************************
;*                                                   **
;*          UNIVERSAL MEASUREMENT SYSTEM             **
;*              CONTROLLER WITH GPIB                 **
;*                                                   **
;*           ABSOLUTE  DATA  DEFINITIONS             **
;*                                                   **
;*  FILE NAME   : UMCEQU.S02                         **
;*  MODULE DEF  : EQUTBL                             **
;*  DATE        : AUGUST 07 1986                     **
;*  AUTHOR      : ILHAN NEDIM ALP                    **
;*                                                   **
;*****************************************************/
;
        NAME    EQUTBL
;
        PUBLIC  USRTPD,USRTMD
        PUBLIC  USRTPC,USRTMC
;
        PUBLIC  PORTA,PORTB,PORTC,PORTCM
;
        PUBLIC  PORT
;
        PUBLIC  TIMER0,TIMER1,TIMER2,TIMERC
;
;***************** USART ADDRESSES ***************
;
USRTPD  EQU     080H
USRTMD  EQU     0A0H
;
USRTPC  EQU     081H
USRTMC  EQU     0A1H
;
;************ 8255 PORT ADDRESSES ***************
;
PORTA   EQU     040H
PORTB   EQU     041H
PORTC   EQU     042H
PORTCM  EQU     043H
;
;********* 74 LS 373 PORT ADDRESS ***************
;
PORT    EQU     050H
;
;********** 8253 TIMER ADDRESSES ****************
;
TIMER0  EQU     0C0H
TIMER1  EQU     0C1H
TIMER2  EQU     0C2H
TIMERC  EQU     0C3H
;
        END
```

```
$PAGEWIDTH(80)
;*************************************************
;*                                               **
;*         UNIVERSAL MEASUREMENT SYSTEM          **
;*             CONTROLLER WITH GPIB              **
;*                                               **
;*                WORKSPACE  RAM                 **
;*                                               **
;*  FILE NAME   : UMCRAM.S12                     **
;*  MODULE DEF  : RAM DEFINITIONS                **
;*  DATE        : JULY 31 1986                   **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
        NAME    DATDEF
;
        PUBLIC  STKTOP,STKEND,STKLEN
;
        PUBLIC  SYSFIL
        PUBLIC  PRTCTR,GPBCTR,SYSCTR
;
        PUBLIC  PRTFIL
        PUBLIC  STAFL,LINPTR,COLPTR
        PUBLIC  DSRCNT,FAILCT,CARFLG
        PUBLIC  LPCNT,TMP2
;
        PUBLIC  OBUFER
;
        PUBLIC  MONFIL
        PUBLIC  RSTATE
        PUBLIC  DIG34,DIG12
        PUBLIC  ONLFLG
        PUBLIC  NXTREG
        PUBLIC  LOWBUF,HIGBUF
        PUBLIC  ADLOW,ADHIGH
        PUBLIC  BUFFER
        PUBLIC  CRFLAG
        PUBLIC  AREG,BREG,DREG,HREG
        PUBLIC  STKPTR
        PUBLIC  PGMCTR
        PUBLIC  BREAK
        PUBLIC  OLDSTK,OLDRET
        PUBLIC  OP1,OP2,OP3
        PUBLIC  OP4,OP5,OP6
        PUBLIC  RECBUF,TRABUF
        PUBLIC  TRKNO,SETNO
        PUBLIC  COMFLG,MSEXIT
;
        PUBLIC  CONV
;
        PUBLIC  EDFIL,EDSTA
        PUBLIC  COLUMN,LINENO
        PUBLIC  KEY,IDLCTR
        PUBLIC  REDSTA,RI
        PUBLIC  FIRST,LAST
        PUBLIC  ENDFLG,JMPFLG
;
        PUBLIC  WBUFER
;
```

```
        PUBLIC  DISBUF

        PUBLIC  INBUF
        PUBLIC  OBUF,OBFCNT
        PUBLIC  LLIST,TLIST
        PUBLIC  VARA,VARB

        PUBLIC  REG

        ORG     0F800H          ; Locate address

----------SYSTEM FILE----------
FIL  EQU     $                  ;10H
CTR: DS      1
CTR: DS      1
CTR: DS      1
SPR: DS      13

----------PRINTER FILE----------
FIL  EQU     $                  ;10H
FL:  DS      6
PTR: DS      1
PTR: DS      1
CNT: DS      1
LCT: DS      1
FLG: DS      1
NT:  DS      1
2:   DS      1
SPR: DS      3

----------PRINTER OUTPUT BUFFER----------
FER: DS      480                ;1E0H

----------MONITOR FILE----------
FIL: DS      48                 ;30H
ATE  EQU     MONFIL+1
S4   EQU     MONFIL+2
12   EQU     MONFIL+3
FLG  EQU     MONFIL+4
REG  EQU     MONFIL+5
BUF  EQU     MONFIL+6
BUF  EQU     MONFIL+7
OW   EQU     MONFIL+8
IGH  EQU     MONFIL+9
FER  EQU     MONFIL+10
LAG  EQU     MONFIL+11
G    EQU     MONFIL+12
G    EQU     MONFIL+14
G    EQU     MONFIL+16
G    EQU     MONFIL+18
PTR  EQU     MONFIL+20
CTR  EQU     MONFIL+22
AK   EQU     MONFIL+24
STK  EQU     MONFIL+26
RET  EQU     MONFIL+28
     EQU     MONFIL+30
     EQU     MONFIL+31
     EQU     MONFIL+32
     EQU     MONFIL+33
     EQU     MONFIL+34
```

```
5       EQU     MONFIL+35
CBUF    EQU     MONFIL+36
ABUF    EQU     MONFIL+37
KND     EQU     MONFIL+38
TND     EQU     MONFIL+39
MFLG    EQU     MONFIL+40
EXIT    EQU     MONFIL+41

----------EDITOR FILE----------
FIL.    EQU     $
STA:    DS      1
LUMN:   DS      1
NEND:   DS      1
Y:      DS      1
LCTR:   DS      1
DSTA:   DS      1
:       DS      1
RST:    DS      1
ST:     DS      1
DFLG:   DS      1
PFLG:   DS      1
SPARE:  DS      5

----------WRITE$BUFFER----------
UFER:   DS      32

----------CONVERTER FILE----------
NV:     DS      6
----------DISPLAY BUFFER----------
SBUF:   DS      33
----------INPUT BUFFER----------
BUF:    DS      32
----------REGISTERS ----------
G:      DS      192
----------OUTPUT BUFFER----------
UF:     DS      28
FCNT:   DS      1
----------VARIABLES----------
RA:     DS      2
RB:     DS      2
----------TALKER/LISTENER LIST----------
IST:    DS      2
IST:    DS      2
----------STACK----------
TKLEN   EQU     128             ;80H

**************************************************
Total           2048            ;800H
**************************************************

TKTOP   EQU     0FF7FH
TKEND   EQU     0FFFFH

ND
```

```
PAGEWIDTH(80)
;*************************************************
;*                                               *
;*      UNIVERSAL MEASUREMENT SYSTEM             *
;*         CONTROLLER WITH GPIB                  *
;*                                               *
;*           DATA DEFINITIONS                    *
;*                                               *
;* FILE NAME   : UMCNVM.S01                      *
;* MODULE DEF. : EEPROM DATA DEFINITIONS         *
;* DATE        : JUNE 11 1986                    *
;* AUTHOR      : ILHAN NEDIM ALP                 *
;*                                               *
;*************************************************

    NAME    NVMDEF

    PUBLIC  LINE

    ORG     8000H

;-------------------------------------------------

    NLINE   EQU     60
    LINESIZ EQU     32
;-------------------------------------------------

    LINE:   DS      NLINE*LINESIZ
;-------------------------------------------------

    END
```

```
PAGEWIDTH(80)
;*************************************************
;*                                               *
;*        UNIVERSAL MEASUREMENT SYSTEM           *
;*            CONTROLLER WITH GPIB               *
;*                                               *
;*                  MONITOR                      *
;*                                               *
;* FILE NAME   : CPUTBL.S05                      *
;* MODULE DEF. : MONITOR TABLE FOR CPU           *
;* DATE        : JULY 31 1986                    *
;* AUTHOR      : ILHAN NEDIM ALP                 *
;*                                               *
;*************************************************

        NAME    CPUTBL

        EXTRN   SYSFIL,PRTFIL
        EXTRN   MONFIL

        PUBLIC  WRIOPS,WRISOP,READPS
        PUBLIC  WRITRK,WRISET,REATRK

        PUBLIC  SGLREC,MLTREC,BUFREC
        PUBLIC  SGLTBL,MLTTBL,BUFTBL

        CSEG

;*************************************************

RIOPS:  DB      0
EADPS:  DB      0
RISOP:  DB      0

RITRK:  NOP
RISET:  NOP
EATRK:  NOP

GLREC:  DB      3,'S',16,'R',16
        DB      'M',48
LTREC:  DB      0
FREC:   DB      0
GL_TBL: DW      MONFIL
        DW      PRTFIL,SYSFIL
LTTBL:  NOP
FTBL:   NOP

;*************************************************

        END
```

```
$NOLIST
;*************************************************
;*                                               **
;*        UNIVERSAL MEASUREMENT SYSTEM           **
;*           CONTROLLER WITH GPIB                **
;*                                               **
;*          GPIB CONTROLLER ROUTINES             **
;*                                               **
;*  FILE NAME   : GPIB.ASM                       **
;*  MODULE DEF. : INCLUDE FILE                   **
;*  DATE        : APRIL 13 1986                  **
;*  AUTHOR      : ILHAN NEDIM ALP                **
;*                                               **
;*************************************************
;
;
        EXTRN   CONIN
;
        PRT91   EQU     0H      ;8291A Base Port #
;
;       Register #0 Data In & Out
;
        DIN     EQU     PRT91+0 ;8291A Data in reg.
        DOUT    EQU     PRT91+0 ;8291A Data out reg.
;
;       Register #1 Interrupt 1 Constants
;
        INT1    EQU     PRT91+1 ;INT Reg. 1
        INTM1   EQU     PRT91+1 ;INT Mask Reg. 1
        BOM     EQU     02      ;8291A BO INTRP Mask
        BIM     EQU     01      ;8291A BI INTRP Mask
        ENDMK   EQU     10H     ;8291A END INTRP Mask
        CPT     EQU     80H     ;8291A Command Pass thru int bit
;
;       Register #2 Interrupt 2
;
        INT2    EQU     PRT91+2
;
;       Register #4 Address Mode Constants
;
        ADRMD   EQU     PRT91+4 ;8291A Address mode register #
        TON     EQU     80H     ;8291A Talk only mode & not listen only
        LON     EQU     40H     ;8291A Listen only & not ton
        TLON    EQU     0C0H    ;8291A Talk & Listen only
        MODE1   EQU     01      ;Mode 1 addressing for device
;
;       Register #4 (Read) Address Status Register
;
        ADRST   EQU     PRT91+4 ;Register #4
        EOIST   EQU     20H
        TA      EQU     2
        LA      EQU     1       ;Listener active
;
;       Register #5 (Write) Auxillary Mode Register
;
        AUXMD   EQU     PRT91+5 ;8291A Auxillary Mode Register
        CLKRT   EQU     23H     ;8291A 3 Mhz clock input
        FNHSK   EQU     03      ;8291A finish handshake command
        SDEOI   EQU     06      ;8291A Send EOI with next byte
        AXRA    EQU     80H     ;8291A auxiliary register A pattern
```

```
HOHSK   EQU     1       ;8291A hold off handshake on all bytes
HOEND   EQU     2       ;8291A hold off handshake on end
CAHCY   EQU     3       ;8291A continuos AH cycling
EDEOS   EQU     4       ;8291A end on EOS received
EOIS    EQU     8       ;8291A output EOI on EOS sent
VSCMD   EQU     0FH     ;8291A valid command pass through
NVCMD   EQU     07H     ;8291A invalid command pass through
AXRB    EQU     0A0H    ;8291A auxiliary register B pattern
CPTEN   EQU     01H     ;8291A command pass through enable

Register #5 (Read)

CPTRG   EQU     PRT91+5

Register #6 Address 0/1 register constants

ADR01   EQU     PRT91+6
DTDL1   EQU     60H     ;Disable major talker & listener
DTDL2   EQU     0E0H    ;Disable minor talker & listener

Register #7 EOS  Character Register

EOSR    EQU     PRT91+7


8292 CONTROL VALUES


PRT92   EQU     20H     ;8292 Base port #

INTMR   EQU     PRT92+0 ;8292 INTRP Mask register
INTM    EQU     0A0H    ;TCI

ERRM    EQU     PRT92+0 ;8292 error mask register
TOUT1   EQU     01      ;8292 timeout for Pass Control
TOUT2   EQU     02      ;8292 timeout for Standby
TOUT    EQU     04      ;8292 timeout for Take Control Sync.
EVREG   EQU     PRT92+0 ;8292 event counter pseudo register
TOREG   EQU     PRT92+0 ;8292 timeout pseudo register

CMD92   EQU     PRT92+1 ;8292 command register

INTST   EQU     PRT92+1 ;8292 interrupt status register
EVBIT   EQU     10H     ;Event counter bit
IBFBT   EQU     02H     ;Input buffer full bit
SRQBT   EQU     20H     ;Seq bit

ERFLG   EQU     PRT92+0 ;8292 error flag pseudo register
CLRST   EQU     PRT92+0 ;8292 controller status pseudo register
BUSST   EQU     PRT92+0 ;8292 GPIB (Bus) status pseudo register
EVCST   EQU     PRT92+0 ;8292 event counter status pseudo register
TOST    EQU     PRT92+0 ;8292 timeout status pseudo register

8292 OPERATION COMMANDS


SPCNI   EQU     0F0H    ;Stop counter interrupts
GIDL    EQU     0F1H    ;Go to idle
RSET    EQU     0F2H    ;Reset
RSTI    EQU     0F3H    ;Reset interrupts
```

```
GSEC    EQU     0F4H    ;Go to standby, enable counting
EXPP    EQU     0F5H    ;Execute parallel poll
GTSB    EQU     0F6H    ;Go to standby
SLOC    EQU     0F7H    ;Set local mode
SREM    EQU     0F8H    ;Set interface to remote
ABORT   EQU     0F9H    ;Abort all operation, clear interface
TCNTR   EQU     0FAH    ;Take control (Receive control)
TCASY   EQU     0FCH    ;Take control asyncronously
TCSY    EQU     0FDH    ;Take control syncronously
STCNI   EQU     0FEH    ;Start counter interrupts

8292 UTILITY COMMANDS

WOUT    EQU     0E1H    ;Write to timeout register
WEVC    EQU     0E2H    ;Write to event counter
REVC    EQU     0E3H    ;Read event counter status
RERF    EQU     0E4H    ;Read error flag register
RINM    EQU     0E5H    ;Read interrupt mask register
RCST    EQU     0E6H    ;Read controller status register
RBST    EQU     0E7H    ;Read GPIB Bus status register
RTOUT   EQU     0E9H    ;Read timeout status register
RERM    EQU     0EAH    ;Read error mask register
IACK    EQU     0BH     ;Interrupt acknowledge

PORT F BIT ASSIGNMENTS

PRTF    EQU     60H     ;Port address of 74LS373
TCIF    EQU     01H     ;Task complete interrupt
SPIF    EQU     04H     ;Special interrupt
OBFF    EQU     08H     ;8292 output (to CPU) buffer full
IBFF    EQU     10H     ;8292 input (from CPU) buffer full
BOF     EQU     01H     ;8291A interrupt line (BO in this case)

GPIB MESSAGES (COMMANDS)

MDA     EQU     1       ;My device address is 1
MTA     EQU     MDA+40H ;My talk address is 1 ("A")
MLA     EQU     MDA+20H ;My listen address is 1 ("!")
UNL     EQU     3FH     ;Universal unlisten
UNT     EQU     5FH     ;Universal untalk
GET     EQU     08      ;Group execute trigger
SDC     EQU     04H     ;Device clear
SPE     EQU     18H     ;Serial poll enable
SPD     EQU     19H     ;Serial poll disable
PPC     EQU     05      ;Parallel poll configure
PPD     EQU     70H     ;Parallel poll disable
PPE     EQU     60H     ;Parallel poll enable
PPU     EQU     15H     ;Parallel poll unconfigured
TCT     EQU     09      ;Take control (pass control)

;*****************************************************************

MACRO DEFINITIONS
```

```
;*************************************************************
;
SETF    MACRO           ;Sets flags on A register
        ORA     A
        ENDM
;
WAITO   MACRO                   ;Wait for 8191A byte to be done
        LOCAL   WAITL
        LOCAL   WAITG
WAITL:  CALL    CONIN           ;Check external interrupt
        JZ      WAITG           ;no interrupt then goon
        RST     0               ;jump RST 0 address to initializtion
WAITG:  IN      INT1            ;Get INT1 status
        ANI     BOM             ;Check for byte out
        JZ      WAITL           ;If not, try again
        ENDM                    ;Until it is
;
;
WAITX   MACRO                   ;Wait for 8232's TCI to go false
        LOCAL   WAITL
        LOCAL   WAITF
WAITL:  CALL    CONIN           ;check external interrupt
        JZ      WAITF           ;no interrupt then goon
        RST     0               ;jump RST 0 address to initialization
WAITF:  IN      PRTF
        ANI     TCIF
        JNZ     WAITL
        ENDM
;
;
WAITT   MACRO
        LOCAL   WAITL
        LOCAL   WAITE
WAITL:  CALL    CONIN           ;check external interrupt
        JZ      WAITE           ;no interrupt then goon
        RST     0               ;jump RST 0 address to initialization
WAITE:  IN      PRTF            ;Get task complete int, etc.
        ANI     TCIF            ;Mask it
        JZ      WAITL           ;Wait for task to be complete
        ENDM
;
;
RANGE   MACRO   LOWER,UPPER,LABEL
                                ;Checks for value in range
                                ;branches to label if not
                                ;in range. Falls through if
                                ;lower (= ( (H)(L) ) (= upper.
                                ;Get next byte.
        MOV     A,M
        CPI     LOWER
        JM      LABEL
        CPI     UPPER+1
        JP      LABEL
        ENDM
;
;
CLRA    MACRO
        XRA     A               ;A=0
        ENDM
;
```

BIBLIOGRAPHY

1. Donald C. Loughry: "What makes a good interface ?" EEE Spectrum, pp 52-57, November 1974

2. David W. Ricci, Gerald E. Nelson: "Standard instrument nterface simplifies system design" Electronics, pp 95-106 ovember 1974

3. Mete Gur: " An evalution of the IEC/IEEE interface us" Electronic Engineering, pp 57-61, September 1978

4. IEEE Standard 488, Digital Interface For Programable nstrumentation, Institute of Electrical and Electronics ngineers, 1978

5. IEEE Standard 728, Recommended Practice for Code and ormat Conventions, Institute of Electrical and Electronics ngineers, 1982

6. Lance A. Leventhal: "8080A/8085 Assembly Language rogramming", Osborne/McGraw-Hill,1978

7. PL/M-80 Programming Manual, Intel Corporation, ocument Number 98-268B

8. Intel Catalog, 1982